An Ontology-based Approach to Web Site Design and Development

Thesis

KNOWLEDGE MEDIA

KMi

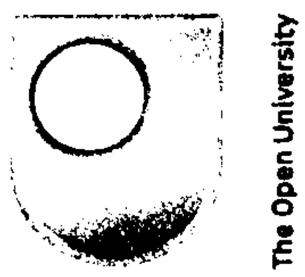
INSTITUTE

An Ontology-based Approach to Web Site Design and Development

Yuangui Lei

Thesis submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy in Artificial Intelligence

Submitted on 29 October 2004



Abstract

Building a data-intensive web site is a complex task. Ad-hoc rapid prototyping approaches easily lead to unsatisfactory results, e.g. poor maintainability and extensibility. The situation becomes even more difficult when customization issues arise and web sites need to present customized views to individual users. To address this problem, a number of model-based approaches have been proposed, which attempt to simplify the design and development of data-intensive web sites. However these approaches suffer a number of limitations, such as relatively little support for the composition of sophisticated user interfaces and the specification of presentation styles and little support for customization design.

In this work we propose and implement an ontology-based approach, OntoWeaver, which provides comprehensive support for the design and development of data-intensive web sites. In particular, OntoWeaver provides a set of ontologies to represent all aspects of data-intensive web sites in a declarative and re-usable format. The declarative nature of the specification of web sites opens up a number of possibilities with respect to intelligent analysis and management. Moreover, OntoWeaver includes providing high level support for developing customized web sites. Finally, it offers a powerful tool suite to support the design and development of data-intensive web sites. In the course of this research, we have also extended OntoWeaver by addressing the issue of integrating web service technology into a high level web site design framework.

Related Publications

Lei, Y., Motta, E., & Domingue, J. (2005), "OntoWeaver: an Ontology-based Approach to the Design of Data-intensive Web Sites", accepted for publication in the Journal of Web Engineering.

Lei, Y., Motta, E., & Domingue, J. (2004), "OntoWeaver-S: Supporting the Design of Knowledge Portals", in Proceedings of the 14th International Conference on Knowledge Engineering and Knowledge Management (EKAW 2004) (Lecture Notes in Computer Science Vol. 3257), October, 2004, Whittlebury Hall, Northamptonshire, UK, pp. 216-230.

Lei, Y., Motta, E., & Domingue, J. (2004), "Integrating Web Services into Data-intensive Web Sites", in Proceedings of the WWW 2004 Workshop on Application Design, Development and Implementation Issues in the Semantic Web, May, 2004, New York, USA.

Lei, Y., Motta, E., & Domingue, J. (2004), "Modelling Data-Intensive Web Sites with OntoWeaver", in Proceedings of the International Workshop on Web Information Systems Modelling (WISM 2004), June 8, 2004, Riga, Latvia, pp. 106-121.

Lei, Y., Motta, E., & Domingue, J. (2003), "Design of Customized Web Applications with OntoWeaver", in Proceedings of the International Conference on Knowledge Capture, October 23-26, 2003, Florida, USA, pp. 54-61.

Lei, Y., Motta, E., & Domingue, J. (2002), "An Ontology-Driven Approach to Web Site Generation and Maintenance", in Proceedings of the 13th International Conference on Knowledge Engineering and Management (Lecture Notes in Computer Science Vol. 2473), October 1-4, 2002, Sigüenza, Spain, pp. 219-234.

Lei, Y., Motta, E., & Domingue, J. (2002), "IIPS: an Intelligent Information Presentation System", in Proceedings of the 7th International Conference on Intelligent User Interfaces, January 13-16, 2002, San Francisco, California, USA, pp. 200-201.

Table of contents

List of figures

List of tables

Chapter 1: Introduction

As the web is becoming the major computing platform for accessing, sharing and exchanging information, the need to develop sophisticated data-intensive web sites which allow the access and management of large sets of data is increasing in domains such as electronic commerce, digital libraries, and distance learning [Atzeni et al., 1998] [Fraternali, 1999]. Building such web sites is however a complex task [Fernandez et al., 1997], which involves not only technical issues, but also organizational, managerial and artistic issues [Morville & Rosenfeld, 1998] [Koch, 1999]. For example, information should be carefully selected, organized, and labelled in order to provide an appropriate presentation; functionalities such as data retrieving, querying, and updating should be provided to allow users accessing and managing the underlying data; and visual appearances such as presentation styles and layouts should be properly designed to provide friendly interfaces. Furthermore, the role of end users needs to be augmented [Brusilovsky, 1996] [De Troyer & Leune, 1998] [Gómez et al., 2000] [Kappel et al., 2000] [Kobsa et al, 2001], which allows web sites to be responsive to the needs of individual users. This makes the situation more difficult, as appropriate customization support needs to be provided which allows the specialization of a general purpose of web site towards the profiles of user groups or individuals.

1.1 Web site design through conceptual web modelling

State-of-the-art web development technologies such as Microsoft's Active Server Pages[1] and JavaSoft's Java Server Pages[2] provide comprehensive solutions to the

[1] http://www.asp.net/ (Accessed 24 June 2005)

[2] http://java.sun.com/products/jsp/ (Accessed 24 June 2005)

extraction and manipulation of dynamic data content from underlying databases. However, ad-hoc rapid prototyping approaches which are driven by these technologies and are adopted in current practice [Murugesan et al., 1999] easily lead to unsatisfactory results, e.g. poor maintainability and poor extensibility [Koch, 1999] [Ceri et al., 2000].

To address this problem, a number of model-based approaches have been proposed, which aim to simplify the design and development process by allowing web sites to be designed at a high level of abstraction and by being able to automatically or semi-automatically produce implementations from high level specifications [Fraternali, 1999] [Koch, 1999] [Retschitzegger & Schwinger, 2000] [Costagliola et al., 2002]. Examples include the Hypertext Design Model (HDM) [Garzotto et al., 1993], the Relationship Management Methodology (RMM) [Isakowitz et al., 1995], the Object-Oriented Hypermedia Design Method (OOHDM) [Schwabe & Rossi, 1998], ARANEUS [Atzeni et al., 1998], HDM-lite [Fraternali & Paolini, 1998], Strudel [Fernandez et al., 1998], the Web Site Design Method (WSDM) [De Troyer & Leune, 1998], the UML-based Web Engineering (UWE) approach [Hennicker & Koch, 2000], OO-H [Gómez et al., 2000], the Web Modelling Language (WebML) [Ceri et al., 2000], OntoWebber [Jin et al., 2001], and Hera [Frasincar et al., 2002].

As will be studied in chapter 2, current approaches distinguish different layers to describe data-intensive web sites and provide models to address each layer accordingly. In particular, they provide comprehensive support for the design of navigation structures. Furthermore, most approaches provide abstract user interface models to address the design of user interfaces explicitly. Finally, most approaches take customization into consideration and provide support for customization. In particular, two solutions have been developed in current approaches. They are composition-level customization which allows the construction of different site views at design time and derivation-level customization which allows the derivation of different site views at run time.

1.2 Thesis motivation

While current web modelling approaches do provide support for the design of web sites at a high level of abstraction, there are still a number of open issues which need to be addressed.

Firstly, one problem common to current approaches is the relatively little support for the specification of site views (i.e. navigation structures and user interfaces). Earlier approaches (e.g., HDM, RMM, OO-HDM, and UWE) are only able to provide methodological guidance for the specification, due to the lack of explicit meta-models. Recent approaches (e.g., WebML, OntoWebber, and Hera) have addressed this problem by providing explicit meta-models. Their meta-models however are defined at a coarse-grained level, which only model typical user interfaces of web pages. No further modelling support is available to allow the adaptation of such typical user interfaces, e.g. removing or adding specific web content. Hence, their support for the construction of complex user interfaces is limited.

Secondly, the problem associated with current approaches is the limited support for the layouts and presentation styles of user interface elements. Web developers have to rely on ad hoc approaches, e.g., Cascading Style Sheets (CSS) [3] and implementation level coding approaches, to achieve a specification. This becomes time-consuming when a web site needs to be rendered in different ways for different purposes, such as different devices, user groups, individuals etc.

The third issue is the limited support for customization both at design time and at run time. Firstly, compositional-level customization can not scale up, as when the number of user groups or individuals grows, the workload of maintaining a large number of site models becomes too heavy. Secondly, the scope of derivation-level customization support is limited, as not all aspects of web sites are available for customization, due to the lack of expressive meta-models for describing the target

---

[3] http://www.w3.org/Style/CSS/ (Accessed 24 June 2005)

web site. Furthermore, specific support for the specification of derivation customization requirements is not available. Finally, some approaches (e.g., WebML and Hera) do not separate customization requirements from other aspects of web sites, thus forcing web developers to anticipate what can be customized at the stage of navigation structure design and user interface design.

The aim of this research project is to address these issues by applying the notion of ontology to model all aspects of data-intensive web sites, including site views, presentation styles, layouts, and customization. An ontology is an explicit formal specification of the terms in a universe of discourse and relations between them [Gruber, 1993] [Neches et al., 1991] [Borst et al., 1995]. By using ontology, web site specification is formalized and can be shared during the life cycle of a web site, thus easing the job of web site maintenance and management, as all components are declarative, re-usable, and represented in shared semantics. Furthermore, the declarative nature of web site specifications opens up a number of possibilities with respect to intelligent analysis and management. Customization and site design critiquing are such examples. Specifically, as the entire site model is available for reasoning, the scope of customization is not limited. Moreover, recommendations can be produced for web developers to improve their design by applying a set of critiquing rules which are able to reason upon the entire site model. Finally, as the provided explicit shared semantics, the target web sites can be picked up by semantic-aware agents and thus benefit from intelligent services (e.g. indexing, searching, and customization) provided by third parties.

The use of ontologies to drive the generation of software tools has been demonstrated in the research area of knowledge acquisition and semantic web portals, where a number of tools have been developed which use domain ontologies to drive the generation of knowledge acquisition tools [Eriksson et al., 1994] [Grosso et al., 1999] [Motta et al., 2000] and semantic web portals [Corcho et al., 2003] [Volz et al., 2003]. This research goes one step further. It applies ontologies to model the target software which is data-intensive web sites, thus facilitating the design and development process. In particular, as clarified above, our goal is to applying

ontologies to address the limitations associated with current web modelling approaches.

1.3 Research approach

The tool described in this thesis, OntoWeaver, employs ontologies to drive the design and development. It models data-intensive web applications by means of i) a comprehensive set of site ontologies which support the conceptual specification of all aspects of data-intensive web sites and ii) a customization framework which supports the specification of customization requirement at design time and the performance of customization at run time.

The site ontologies include *a site view ontology* which models site views (i.e. navigation structures and user interfaces) of data-intensive web sites and *a presentation ontology* which captures the features of the presentation styles and layouts of site view components:

- The site view ontology provides fine-grained modelling support for user interfaces and navigation structures of the target web site. It addresses the first issue of current approaches described in section 1.2. Unlike current approaches which only address typical user interface elements of web pages, the site view ontology addresses the specification of atomic user interface elements, generic composite user interface elements, as well as typical user interface elements. It realizes a composition mechanism and allows web developers to express complex user interfaces according to their own requirements. At the same time, it addresses static user interface elements, whose content is defined at design time, as well as dynamic user interface elements which deal with the back-end data sources dynamically at run time.
- The presentation ontology provides high level support for the specification of layouts and presentation styles for user interface elements. It addresses the second issue of current approaches. In particular, it allows web developers expressing complex layouts and presentation styles at the conceptual level. Web developers no

longer need to encode the specification into web page implementations, like they have to in other approaches.

The customization framework addresses the third issue by means of exploiting the advantage of the declarative specification of web sites, separating the specification of customization requirements from other aspects of web sites, providing support for the specification of customization requirements at design time and for the execution of customization at run time. Firstly, as all user interface elements and their presentation instructions are specified declaratively, the entire site model is available to customization. Secondly, the customization framework separates the specification of customization from other aspects of the target web site, thus enabling the web site design process to be more flexible. Web developers do not need to anticipate what can be customized at the stage of site view design, like they have to in approaches like WebML, WUML [Kappel et al., 2001, 2002], and HERA. Finally, OntoWeaver provides comprehensive support for the specification of customization requirements. It offers a customization rule model to support the construction of customization rules. Thus, web developers do not have to rely on ad-hoc approaches to augment individual's roles into site specifications.

As mentioned earlier, building data-intensive web sites is a complex task, which involves not only technical issues, but also organizational, managerial and artistic issues. Different roles are thus involved in the design process. Domain experts design models that can abstract data of a specific problem domain. Information architects structure, organize, and label information. Site designers put the design into practice and carry out customization design. Layout designers make decisions on presentation styles and layouts. Web administrators manage and maintain the web site.

The OntoWeaver approach aims to provide comprehensive support for these different roles to achieve their tasks. Specifically, it supports information architects to structure, organize, and label information at the conceptual level by means of the site view ontology, layout designers to specify presentation styles and layouts at a high level of abstraction by means of the presentation ontology, site designers to specify customization requirements by means of the customization framework, and

web administrators to maintain the target web site. These facilities will be illustrated in chapter 7 by building a concrete data-intensive web site. The web site design process in OntoWeaver involves the following steps:

- Requirements collection and analysis which identify the objectives of the target web sites, data characteristics, user categories, user requirements, and so on.
- Domain ontology design which develops a data model to abstract the problem domain.
- Site view design which makes use of the site view ontology to specify site structures and detailed user interfaces for page nodes.
- Presentation design which makes use of the presentation ontology to express presentation styles and layouts for site view elements in the site view model.
- Customization design which identifies and specifies customization requirements in terms of user ontology and customization rules.

1.4 Thesis contributions

The main characteristics of OntoWeaver comprises i) using ontologies to drive the design and development of data-intensive web sites, ii) providing a comprehensive set of ontologies to support the specification of all aspects of web sites, and iii) providing a customization framework to support the specification of customization requirements at design time and the delivery of customization support at run time. Furthermore, we have developed an OntoWeaver tool suite which facilitates different tasks involved in web site design and an OntoWeaver extension called OntoWeaver-S which addresses the issue of integration of web services with data-intensive web sites. Hence, the major contributions of this research include the following components:

- The site view ontology
- The presentation ontology
- The customization framework
- The OntoWeaver tool suite
- OntoWeaver-S

1.5 Thesis structure

In chapter 2 we review the key approaches which have been proposed for addressing the design and development of data-intensive web sites. We focus on the capabilities of these approaches on supporting a number of key modelling activities, including domain modelling, site view modelling, presentation modelling and customization modelling.

In chapter 3 we investigate current customization techniques on user modelling, user information acquisition, and customization delivery. We also examine tools and approaches which support customization design for web applications. Our purpose is to motive our work on developing a generic customization framework which on the one hand is able to provide comprehensive support for the specification of customization requirements at design time and on the other hand is able to deliver sophisticated customization at run time.

In chapter 4 we present an overview of the OntoWeaver design framework. We begin by re-stating the limitations of current high level design approaches learned from the study carried out in chapter 2. We then clarify the requirements placed upon high level design approaches to address these issues. Thereafter, we present an overview of our approach with respect to the research goals, assumptions, restrictions, and major contributions.

In chapter 5 we focus on one major component of the OntoWeaver design framework – the site ontologies. We first clarify the two components of the site ontologies – the site view ontology and the presentation ontology and in particular illustrate how they provide comprehensive support for the specification of data-intensive web sites. We then describe the generation of web site implementations from ontologies. Thereafter, we relate our work on the site ontologies to existing approaches on conceptual web modelling, user interface modelling, presentation modelling, using ontologies to drive web site generation, and reference models.

In chapter 6 we describe the OntoWeaver customization framework which exploits a user model, a customization rule model, and a declarative site model to enable the

specification of customization requirements at a conceptual level. We explain the customization framework components in detail and carry out a case study on employing the customization framework to build a conference paper review system.

In chapter 7 we present the OntoWeaver tool suite which provides comprehensive support for the building and maintenance of data-intensive web sites powered by the OntoWeaver modelling approach. We begin by describing the facilities provided by the OntoWeaver tool suite. We then illustrate how these facilities support different design roles to achieve their tasks by building an example data-intensive web site. Thereafter, we illustrate the benefits gained by using site ontologies to represent web site specifications. Next, we compare the facilities provided by OntoWeaver to the facilities provided by other conceptual web modelling approaches with respect to the support for web site specification, customization design, web site design critiquing, and web site maintenance.

In chapter 8 we describe our work on extending OntoWeaver to OntoWeaver-S which aims to provide high-level support for the design and development of web service powered data-intensive web sites. We begin by describing the research motivation. We then present how to move OntoWeaver to OntoWeaver-S and how OntoWeaver-S supports the design of data-intensive web sites which allow access to both the underlying domain data and the specified web services. Next, we describe the implementation of OntoWeaver-S and the related work.

Finally, in chapter 9 we summarize our contributions in this work and outline the future research directions concerning this work.

Chapter 2: Web site design through conceptual modelling

In this chapter we review the key approaches which have been proposed to simplify the design and development of web sites by allowing them to be specified at a high level of abstraction. We begin by clarifying a set of criteria which should be satisfied for high level design approaches to provide appropriate support for web site design and development. We then present a brief overview for each key approach. Thereafter, we employ the clarified criteria to go through the approaches to investigate their fulfilment on these criteria. Finally, we summarize the open issues of current approaches which emerge from the review.

2.1 Review criteria

Web sites are *data-intensive* when their primary goal is to allow users to access large amounts of data [Ceri et al., 1999a]. They can be characterized by the following major design dimensions:

- *Domain data structure,* which describes the information that is to be managed and presented by the target web site.
- *Navigation,* which concerns the facilities that allow end users to browse and navigate across the target web site.
- *User interface,* which describes the composition structure of the contents of web pages that allow dynamic access to underlying data sources.
- *Presentation*, which expresses the look and feel (i.e. presentation styles and layouts) of user interface elements.
- *Customization*, which describes the way to specialize a general purpose of web site towards the profiles of user groups or individuals. Please note that we do not necessarily distinguish the words customization, personalization, and adaptation,

as all of them refer to the process of specialization of web sites towards user groups or individuals.

This architecture is derived from the architectures described in [Fraternali, 1999] [Ceri et al., 1999a] [Kappel et al., 2000]. Unlike the previous ones which tightly couple contents of web pages and their presentation styles and layouts together, this architecture distinguishes the design of presentation styles and layouts from the design of web page contents, thus allowing different presentation instructions to be easily specified for the same site view (i.e. contents and navigation structures) for different purposes, e.g., different devices and different user groups. Furthermore, like the one described in Kappel et al (2000), this architecture takes customization as an important design dimension, as web sites offer information which is potentially interesting to a large number of audiences [De Bra et al., 2003] and web sites need to be responsive to the individual needs of end users [Brusilovsky, 1996] [De Troyer & Leune, 1998] [Rossi et al., 2001].

To provide appropriate support for the design and development of data-intensive web sites, high level design approaches should provide comprehensive support for the modelling of these design dimensions namely, domain modelling, navigation modelling, user interface modelling, presentation modelling, and customization modelling. Otherwise, web developers will have difficulty specifying web sites at the conceptual level without turning to ad-hoc programming approaches. In particular, with only coarse-grained level support for navigation modelling and user interface modelling, web developers will have difficulty creating site views (i.e. navigation structures and user interfaces) which meet requirements of individual web sites, as they can not be simply defined by means of a fixed number of coarse-grained primitives without further support for adjustment. Furthermore, without appropriate modelling support for presentations styles and layouts, web developers will have to rely on ad-hoc approaches, such as cascading style sheets (CSS) and implementation level coding approaches, to achieve a specification. This is a time consuming process in particular when the same site view needs to be rendered into different presentations for different purposes. Finally, web developers will have to rely on ad-hoc approaches to augment users' customization requirements into web site

specifications or even into web page implementations to realize customization when there is no comprehensive customization modelling support available.

2.2 Model-based web site design approaches

In this section, we briefly describe the key high-level approaches to web site design. These approaches include the Hypertext Design Model (HDM) [Garzotto et al., 1993], the Relationship Management Methodology (RMM) [Isakowitz et al., 1995], the Object-Oriented Hypermedia Design Method (OOHDM) [Schwabe & Rossi, 1998], ARANEUS [Atzeni et al., 1998], HDM-lite [Fraternali & Paolini, 1998], Strudel [Fernandez et al., 1998], the Web Site Design Method (WSDM) [De Troyer & Leune, 1998], the UML-based Web Engineering (UWE) approach [Hennicker & Koch, 2000], OO-H [Gómez et al., 2000], the Web Modelling Language (WebML) [Ceri et al., 2000], OntoWebber [Jin et al., 2001], and Hera [Frasincar et al., 2002].

2.2.1 The Hypertext Design Model (HDM)

The Hypertext Design Model (HDM) is one of the first design methods that have been proposed to support the design of hypertext applications. It is based upon the Entity Relationship (E-R) model [Chen, 1976] and extends the concept of entity by introducing new primitives such as *perspective* which describes the concept of having different presentations for the same content, *unit* which models components that is associated with a specific perspective, and *link* which describes navigation paths. Thus, HDM allows the description of overall classes of information elements and navigational structures of hypertext applications. HDM distinguishes two layers to model a hypertext application. They include *a hyperbase layer* which represents the content of the hypertext application and *an access layer* which provides navigation facilities.

Customization is explicitly pursued in HDM by means of the concept *perspective* which abstracts different presentations for the same content. For example, the same content can have different language perspectives, e.g., English and French. However, HDM relies on developers to decide how to use the notion of perspective to present

information in different ways. JWeb [Bochicchio et al., 1999] is a HDM-based tool for the design and development of web sites.

2.2.2 The Relationship Management Methodology (RMM)

The Relationship Management Methodology (RMM) addresses the design of hypermedia applications by embedding the design concepts of HDM into a structured methodology. It proposes a model called Relationship Management Data Model (RMDM) and a set of guidelines to support the design and development.

RMM distinguishes seven steps to the design of hypertext applications. They are the E-R design which is describes information of the application domain by means of the Entity Relationship (E-R) model, the slice design which groups entity attributes as slices for presentation, the navigational design which specifies navigation structures through the provided access primitives, the conversion protocol design which converts components of the RMDM diagram into physical objects in the target application, the user interface design which involves the design of layouts for every element appear in the specification, the run time behaviour design which decides whether node contents and link endpoints are to be built at run-time according to the volatility and the size of the domain, and finally the construction and testing.

RMM focuses on the design of first three steps. It relies on the E-R model to achieve the task of the first step. At the same time, RMM provides a construct called *slice* to facilitate the slice design by splitting an entity into meaningful slices and organizing these into a hypertext network, and a set of *access primitives* to support the specification of navigation structures. Example access primitives include *uni-directional* and *bi-directional links* which specify access between slices of an entity, *index* which lists entity instances and provides direct access to each listed item, *guided tour* which implements a linear path through a collection of items allowing the user to move either forward or backward on the path, *the grouping primitive* which is a menu-like mechanism that enables access to other parts of a hypermedia documents, and *condition* which filters instances of an entity. RMC [Diaz et al.,

1995] is an RMM-based tool for supporting the design and implementation of data-intensive web sites.

2.2.3 The Object-Oriented Hypermedia Design Method (OOHDM)

The Object-Oriented Hypermedia Design Method (OOHDM) makes use of an object-oriented approach for web site design. The development occurs as a four-activity process in OOHDM:

- *Conceptual design*, which employs the object-oriented modelling principles to represent domain models in terms of *classes* and *relationships*. Classes are used to derive page nodes while relationships are used to derive links in the phase of navigation design.

- *Navigational design*, which constructs navigation structures. OOHDM provides comprehensive support for navigational design. Firstly, OOHDM provides a set of *navigation classes*, such as *nodes*, *links*, *indices*, and *guided tours* to support the construction of navigation objects. Nodes represent logical views on classes defined in domain data models. Links connect nodes together to enable navigation. Secondly, OOHDM introduces the concept of *navigational context* to describe navigation structures by grouping navigation objects into different contexts for the purpose of navigating them in different contexts.

- *Abstract interface design*, which specifies user interface objects that users will perceive. OOHDM relies on the *Abstract Data View* design approach [Cowan et al., 1995] to specify user interfaces. Specifically, OOHDM maps the navigational constructs to *Abstract Data Objects* (ADO), which are constructs for composing user interfaces. Moreover, OOHDM employs *Abstract Data Views* (ADV) to specify the behaviours and the key events associated with user interface objects.

- *Implementation*, which maps conceptual objects, navigation objects, and user interface objects to the particular run-time environment targeted.

The OOHDM approach focuses particularly on the navigation design and the abstract interface design. The mechanism of separating the navigational model from the conceptual model allows developers to take user profiles and tasks into account and build different navigational structure over the same conceptual model. Likewise, different user interfaces can be constructed over the same navigation structure. OOHDM-Web [Schwabe et al., 1999] is a tool that allows web site generation based upon the OOHDM approach.

2.2.4 ARANEUS

The ARANEUS approach extends HDM, RMM, and OOHDM by providing support for automating the implementation and maintenance of the target web site. It distinguishes two interconnected processes namely, the database design process and the hypertext design process. Each design process is further divided into a conceptual phase and a logical design phase. ARANEUS provides a logical data model called *ADM* (standing for ARANEUS Data Model) which represents an abstract description of actual web pages. Furthermore, it provides a language called *PENELOPE* which supports the automatic generation of web pages from databases.

ARANEUS defines a number of navigation constructs which facilitate the specification of navigation models. Furthermore, it provides a set of user interface constructs which enable the specification of user interfaces for web pages. However, these user interface constructs are defined at a coarse-grained level, as basic user interface elements (e.g., output elements and command elements) are missing. For example, the user interface elements, which are specified by means of the provided constructs, e.g., *lists* and *forms*, can not be adjusted and extended, due to the lack of further modelling support.

2.2.5 HDM-lite

HDM-lite specifies data-intensive web sites by means of a *hyperbase schema* which describes the underlying data model of the target web site, an *access schema* which specifies navigation structures, and a *presentation schema* which specifies the

appearance of application pages. In particular, the HDM-lite approach provides means to support the specification of user interfaces and their visual appearances. The basic unit of presentation is the construct *page* whose appearance is logically specified in two levels: 1) the layout level describes page layout: the layout of web pages is modelled as a grid whose cell may contain presentation elements, and ii) the element level specifies elements appeared in page grid. HDM-lite provides pre-defined built-in elements to support the visualization of the main concepts of the hyperbase and access schemas.

The Autoweb system [Fraternali and Paolini, 1998] is a tool based on the HDM-lite approach. The system comprises a case tool which supports the input of HDM-lite conceptual schemas and their transformation into relational structures and an automatic page generation tool which automatically generates application pages from web site specifications.

2.2.6 Strudel

Strudel applies concepts from database management systems to the process of building web sites. The design process in Strudel mainly comprises three steps: accessing and integrating data available in the target web site, building site structures i.e. specifying data in each page and links between pages, and generating HTML implementations of web pages.

The architecture of Strudel comprises a semi-structured data model which supports heterogeneous data sources by providing wrappers that convert data in external sources into the Strudel graph format, a data mediator which supports data integration by providing a uniform view of all underlying data, a query processor which constructs a web site specification, and a site generator which generates HTML pages by associating an HTML template with each object in the site graph.

Strudel provides a declarative query language called *StruQL* which supports the specification of site graphs (i.e. site structures) by means of defining queries over the

domain data graphs and a template language which supports the design of templates that compile site graphs to HTML pages.

2.2.7 The Web Site Design Method (WSDM)

The Web Site Design Method (WSDM) uses the requirements of the intended users to drive the process of web site design. End users are classified into audience classes, and the available data and navigational structures are modelled from the point of the different audience classes. As a result, the target web sites are tailored according to their end users, and therefore their usability is greatly enhanced.

The design process in WSDM starts with defining mission statements which express the purposes of the target web site. Web developers then model audiences, which produces a set of audience classes together with an informal description of their requirements. Thereafter, developers create domain data models and navigation models for members from different audience classes. The fourth step concerns structures and the 'look and feel' of web pages, which aims to design page structures. The final step realizes the target web site using the chosen implementation environment.

While the WSDM approach provides comprehensive support for addressing requirements of users in the design of web sites, it results in huge amounts of redundant information when there are a large number of user groups involved in the target web site, as the customization strategy in WSDM is creating a domain model and a site view for each user group or each individual user. This makes the target web site very hard to maintain.

2.2.8 The UML-based Web Engineering (UWE)

The UML-based Web Engineering (UWE) approach is proposed upon the basis of RMM, OOHDM, and WSDM. It uses some graphical elements from RMM, e.g. *index* and *guided tour*. It separates the construction of conceptual, navigation, and

presentation models that stems from OOHDM, and continues with the user-centred approach of WSDM.

UWE employs a use case model to capture the requirements of web sites. The domain modelling takes use cases into consideration and produces a conceptual model in terms of UML class diagrams. The navigation modelling in UWE is addressed by means of a navigation space model which specifies which objects can be visited by navigating through the web application and a navigation structure model which defines how the navigation is supported by access elements, e.g., *indexes*, *guided tours*, *queries* and *menus*.

The UWE approach addresses the design of user interfaces by means of a set of user interface constructs, including a set of navigational construct based elements which are mapped from the navigational constructs, a set of primitive user interface elements which present text, image, form, button, video and audio, and a set of composition constructs which specify composite user interface elements for holding sub elements (examples include *frameset*, *frame*, and *window*). However, although user interface constructs are provided, there are no explicit meta-models available, which provide formal definition for these constructs. Hence, they can only be used as guidance to support the design of user interfaces. The specifications of target web sites are therefore not entirely declarative.

Customization modelling in UWE is supported by means of a user model which serves as a meta-model for constructing user profiles and an adaptation model which comprises a set of adaptation rules to specify the adaptation conditions, actions, and principles for upgrading user models and adapting links and contents of web pages. This customization framework separates the specification of customization requirements from other aspects of web sites. Therefore, web developers do not have to anticipate what can be customized at the stage of specifying navigation structures and user interfaces, which makes the design approach more flexible. However, as discussed above, the specification of web sites in UWE is not declarative. As a result, the customization support is limited.

In addition to the feature of separating customization requirements from other aspects of web sites, the UWE approach covers most aspects of web applications, including use cases, content, navigation, presentation, adaptation, and end users, and furthermore provides guidelines to support the design and development of each design aspect.

2.2.9 OO-H

OO-H is a user-centred approach, which makes use of user navigation requirements to drive the design of web sites. It extends the traditional object-oriented modelling approach with two new diagrams namely, *a Navigation Access Diagram* (NAD) and *an Abstract Presentation Diagram* (APD). While the *NAD* captures navigational structures of web applications, the APD gathers the concepts related to presentation. In particular, the APD allows developers to make some decisions on how to present web pages by choosing navigation patterns and page patterns. In addition, the OO-H approach addresses special features of web applications by means of frameworks and patterns that are gathered in an OO-H pattern catalogue.

Regarding support for customization, OO-H takes user requirements into consideration from the phase of the navigation design. Developers can define different NADs over a single domain class diagram. Each NAD instance reflects the information, services, and required navigation paths for the associated user. Moreover, developers can define different APDs for the same NAD. However, as WSDM, this customization support is efficient only when there are a very limited number of user types involved in the target web applications. When the number grows bigger, the workload of designing, managing, maintaining, and deploying large number of navigation models and presentation models will become too heavy.

2.2.10 The Web Modelling Language (WebML)

The Web Modelling Language (WebML) is a conceptual modelling language, which distinguishes six perspectives to support the high level specification of data-intensive web sites. They are the *structural model* which expresses the data structure of the

site, in terms of the E-R model, the *derivation model* which extends the structural model with redundant data to enable the adaptation according to different user requirements, the *composition model* which concerns the definition of pages and their internal organizations, the *navigation model* which describes the links between pages and content units to form hypertext structures, the *presentation model* which uses style sheets to express the layout and graphic appearance for web pages, independently of the output device and of the rendition language, and finally the *personalization model* which specifies the requirements of personalization.

WebML models views of data-intensive web sites by means of a set of navigational constructs and a set of user interface constructs which model the typical user interface elements of data-intensive web sites for data presentation, data acquisition, and data querying. These user interface constructs however can only provide coarse-grained level support for specifying web pages. For example, WebML views data units which publish dynamic data content of the backend databases as atomic user interface elements which can not be further composed of. However, they should consist of a number of components, e.g. texts, hyperlinks and images. Thus, more fine-grained constructs should be proposed to allow data units to be composed and manipulated.

Customization in WebML is supported at three levels: i) at the composition level, different site views can be defined for pre-defined user groups, ii) at the derivation level, site designers define derived concepts (e.g. entities, attributes) whose definitions depend on user-specific data, and iii) at the business rule level, rules can be specified in terms of event-condition-action for e.g. classifying users into pre-defined user groups, managing user profiles, assigning the users to user groups based on dynamically collected information (e.g., the purchase history, or the access device), and updating the site content (e.g., inserting new offers matching users' preference).

Torrisoft is an environment for the computer-aided design of web sites using the WebML approach. The WebRatio Site Development Studio[1] is a commercial web site design tool, which is based on WebML.

2.2.11 OntoWebber

OntoWebber is an ontology-based approach, which uses ontologies to drive the design and development of data-intensive web sites. It provides an explicit meta-model called site ontology, which models navigation, content, and presentation of data-intensive web sites. The major constructs include *card*, *page*, *link*, and *site-view graph*. A card is the basic unit to compose pages which correspond to physical web pages. A link connects cards together to form navigation structures. A site-view graph is a graphical representation of three aspects of a site view, i.e., navigation, content, and presentation. To support the modelling of typical user interfaces and their layouts, a set of card and layout primitives are provided in OntoWebber.

OntoWebber distinguishes a number of models to enable the specification of target web sites, including a *domain model* which represents common concepts and relationships in the domain, a *site view-graph* which describes a simplified conceptual model of the target web application, a *presentation model* which provides primitives to define presentation styles and layouts for web pages, a *personalization model* which describes user information, including interest, capacity, and request etc, and a *site maintenance model* which focuses on content maintenance from data management point of view.

The architecture of OntoWebber comprises an integration layer which resolves syntactic differences between different distributed heterogenous data sources by converting source data into RDF, an articulation layer which resolves the semantic differences between the converted source data by relating them to the OntoWebber

---

[1] http://www.webratio.com (Accessed 24 June 2005)

reference ontology, a composition layer which creates site specifications, and the generation layer, which generates a browsable web site.

Unlike other approaches, OntoWebber explicitly addresses the maintenance issue. It identifies two types of maintenance rules. One is the user-oriented maintenance rules which target the maintenance of end users' site views. The other is the site-oriented rules which allow site administrators to perform maintenances. Furthermore, OntoWebber addresses the validation of web site specifications in the process of web site generation by providing a set of constraints to verify the navigation, content, and presentation aspects.

OntoWebber also provides two levels of support for customization. First, at the coarse-grained level, different site views can be defined for different user groups and individuals. Second, at the fine-grained level, a personalization schema is provided in terms of the class *User* to i) describe information about end users, such as age, preferred browser type, etc, ii) specify user's view on the three models of the site view, and iii) define triggers which will be fired when certain conditions are satisfied.

Although OntoWebber addresses the issues of site maintenance, validation, and customization, it does not provide any modelling support for the specification of maintenance rules, validations constraints, and user's view and triggers in customization. Furthermore, the modelling support for user interfaces is not comprehensive, as only a fixed number of primitives are provided. Hence, the user interfaces it can describe are limited. The OntoWebber system [Jin et al., 2002] provides a software environment to allow the constructrion of data-intensive web applications using the OntoWebber approach.

2.2.12 Hera

The Hera approach is proposed upon the basis of the RMM approach. It distinguishes several steps to design a data-intensive web site. They include the conceptual design which produces a conceptual model describing data structures of the problem

domain, the application design which focuses on the specification of the presentation and navigation aspects, the adaptation design which annotates the application model by adding appearance conditions to components of the application model, and the presentation design which maps models produced in the previous steps to a presentation diagram which describe how to render the target web site.

The HERA approach extends the concept *slice* of RMM by introducing the property *slice-ref* to the class *Slice*. As a result, it allows developers to specify compositional structures for slices which are major components of web pages in Hera. Furthermore, the HERA approach introduces adaptation design into the design process of web sites. The adaptation design is carried out by adding user specific conditions to components of the application model to compute their visibility according to profiles of end users.

2.3 A comparison

In section above, we have presented a brief overview of each key web modelling approach. In this section, we compare their support on the modelling activities which as described earlier in section 2.1 are crucial for providing appropriate support for the design and development of data-intensive web sites.

Although current approaches share a similar goal of offering high level support for web site design and development, they have different focuses. Some approaches, e.g., ARANEUS, Strudel, WSDM, UWE, OO-H, WebML, and OntoWebber, try to address many aspects in the design and development process, whereas others, e.g., HDM, HDM-lite, and RMM, concentrate on some of them. For example, while HDM mainly focuses on the navigation structure design, the UWE approach aims to cover as many design aspects as possible. Furthermore, conceptual web modelling approaches are always under constant development, because web sites, their target, are evolving all the time. These features make comparing web modelling approaches a very difficult task.

In this section we present a comparison of the approaches from the perspective of high level support for the *specification* of web sites as clarified in section 2.1. Table 2.1 shows the comparison. We used four categories to classify support, including i) full support (■ ■ ■), ii) fair support (□ ■ ■), iii) poor support (□ □ ■), and iv) none support at all (□ □ □). In the following sub sections, we will explain the comparison result in more detail.

Table 2.1 A comparison of the reviewed approaches on their modelling support for the key design activities of data-intensive web sites

Dimensions / Approaches	Domain modelling	Navigation modelling	User interface modelling	Presentation modelling	Customization modelling
HDM	□ ■ ■	□ ■ ■	□ □ □	□ □ □	□ □ ■
RMM	■ ■ ■	□ ■ ■	□ □ ■	□ □ □	□ □ ■
OOHDM	■ ■ ■	□ ■ ■	□ □ ■	□ □ □	□ □ ■
ARANEUS	■ ■ ■	□ ■ ■	□ □ ■	□ □ □	□ □ ■
HDM-lite	■ ■ ■	□ ■ ■	□ □ ■	□ □ □	□ □ ■
Strudel	■ ■ ■	□ ■ ■	□ ■ ■	□ □ □	□ □ ■
WSDM	□ ■ ■	□ ■ ■	□ □ □	□ □ □	□ ■ ■
OO-H	■ ■ ■	□ ■ ■	□ ■ ■	□ □ □	□ ■ ■
UWE	■ ■ ■	□ ■ ■	□ □ ■	□ □ □	□ ■ ■
WebML	■ ■ ■	■ ■ ■	□ ■ ■	□ □ □	□ ■ ■
OntoWebber	■ ■ ■	■ ■ ■	□ ■ ■	□ ■ ■	□ ■ ■
Hera	■ ■ ■	■ ■ ■	□ ■ ■	□ □ □	□ ■ ■

2.3.1 Domain modelling

Domain modelling concerns the semantics of the underlying domain databases of data-intensive web sites. It has been explicitly addressed by most approaches. The support is achieved by allowing the abstraction of domain data structures and by allowing the specification of navigation structures and user interfaces for accessing the underlying domain data. In particular, in WebML domain data content is not only readable, but also manageable, as WebML has defined a set of operation constructs to support the management of the underlying domain data.

Other approaches, including HDM and WSDM, on the other hand do not support dynamic access to underlying domain data, as they target static web sites, where information is pre-defined at design time and the presentation of dynamic data content is not an issue. Nevertheless, they separate domain data from the specification of other aspects of the target web site.

In addition to allowing the presentation of domain data represented in a pre-defined data structure, approaches like Strudel and OntoWebber take things one step further. They introduce the facility of data integration into the design and development loop, thus supporting the presentation of heterogeneous data sources represented in different schemas. Such facility can be re-used in other approaches.

2.3.2 Navigation modelling

Navigation modelling concerns the navigation structures and navigation behaviours of the target web sites. A number of comprehensive methods and primitives have been proposed to support this modelling activity. Examples include the access primitives (e.g., *index*, *guided tours*, *uni-directional links*, and *bi-directional links*) in RMM and Hera; the navigational classes in OOHDM; the navigational conceptual model (NCM) in ARANEUS; the navigation model in UWE, WebML, and OntoWebber; and the Navigation Access Diagram (NAD) in OOH. However, except most recent approaches, including WebML, OntoWebber, and Hera, other approaches do not provide explicit meta-models to support the specification at the conceptual level. As a consequence, they can only provide methodological support for the creation of navigation structures.

2.3.3 User interface modelling

User interface modelling concerns the composition of contents of web pages. It has been addressed in many approaches. For example, OOHDM employs an external approach to describe abstract user interface objects of web applications; ARANEUS makes use of the logical data model *ADM* to represent an abstract description of actual web pages; Strudel relies on the template language to support the design of

pages; UWE provides an abstract user interface model; and finally approaches like OOH, WebML, and OntoWebber propose comprehensive primitives to describe typical user interfaces of data-intensive web sites, such as the user interfaces for data presentation, data acquisition, and data querying.

However, as the situation discussed above in the navigation modelling, except the most recent approaches (including WebML, OntoWebber, and Hera), other approaches do not provide explicit meta-models which allow user interfaces to be specified at the conceptual level. Thus, they can only be used as guidance to support the design of user interfaces.

For the most recent approaches, even though comprehensive coarse-grained primitives are proposed to model typical user interfaces of web pages and explicit meta-models are provided to support the conceptual specification, no further modelling support is available to allow the adaptation of typical user interfaces. For example, no modelling support is provided to address *atomic* user interface elements, which are components of typical user interfaces of data-intensive web sites, such as elements that i) present static information, ii) present dynamic information, iii) allow input from end users, and iv) allow the invocation of available services. As a consequence, web developers are only able to express a fixed number of typical user interfaces in terms of the provided primitives. The creation of complex user interface is out of reach at the conceptual level.

2.3.4 Presentation modelling

The modelling of presentation styles and layouts has not been fully addressed in current approaches. Most approaches rely on external approaches (e.g., style sheets) to realize a specification. This is mainly because the components of web pages are not entirely represented in a declarative format and thus not all available for associating presentation instructions at the conceptual level, due to the lack of explicit expressive meta-models for user interface specification. As a consequence, web developers still need to do a lot of low-level work to create complex presentation styles and layouts for web pages.

HDM-lite partially addresses the specification of layouts for web pages by using grid to position sub-elements of page nodes. OntoWebber proposes a set of layout primitives (e.g., *flow layout* and *grid layout)* to describe typical layouts of user interface elements. However, these primitive do not support the expression of complex layouts, e.g., the composition of different layouts.

2.3.5 Customization modelling

As web sites offer information which is potentially interesting to a wide range of audiences, they are required to be capable of presenting customized views to individual users. This requirement has been addressed in current approaches.

Firstly, *composition-level* customization is made possible by all the approaches mentioned in this section by strictly separating domain data models from site view models, thus allowing different site view models to be defined over the same domain model for different user groups or individuals. Approaches like WSDM and OOH exploit this advantage and provide comprehensive customization support for user groups and individuals by using the requirements of the target users to drive the design and development of web sites. End users are classified into user groups. Navigation structures and user interfaces are then designed for each user group. As discussed earlier, one major problem of this method is that it cannot scale up. As the number of user groups grows, the workload of designing and maintaining a large number of site models becomes too heavy.

Secondly, *derivation-level* customization is developed in approaches like UWE, WebML, OntoWebber, and Hera, which allows the derivation of customized views for individual users according to their profiles and customization requirements at run time. It provides dynamic customization support. Two major solutions have been developed. One is the solution employed in approaches like WebML, OntoWebber, and Hera, which annotates the specification of the target web site with user information. The other solution is the one adopted in UWE, which relies on rules to specify customization requirements. However, this kind of customization support is limited in current approaches, as not all aspects of web sites are available for

customization due to the lack of expressive meta-models for describing the target web site. Furthermore, specific support for the specification of customization requirements is not available. For example, in annotation-based approaches, no specific models are available to support the specification of user annotation and the association of annotations with site specifications. Analogously, in rule-based approaches, although generic modelling support such as UML is provided, no specific modelling support is available to allow, e.g., the definition of customization conditions and adaptation actions. Finally, the annotation-based approach does not separate the specification of customization requirements from other aspects of data-intensive web sites. As a consequence, web developers have to anticipate what can be customized at the stage of navigation structure design and user interface design.

Approaches like HDM, RMM, OOHDM, ARANEUS, and HDM-lite do not provide explicit customization support. They rely on web developers to resolve the issue of customization design by using ad hoc approaches to create user models and assign different site views for different user groups and individuals.

2.4 Open issues

In summary, current approaches typically distinguish three layers to describe data-intensive web sites, including *a data layer* which describes the underlying domain data of the target web site, *a navigation structure layer* which describes the navigation structures of the target web site, and *a user interface layer* which describes the user interfaces of the target web site. They provide models to address each layer accordingly. Firstly, current approaches provide comprehensive support for the design of navigation structures. Secondly, most approaches address the design of user interfaces explicitly or implicitly. Finally, most approaches take customization into consideration and provide customization support for the target web site to a variety of levels. Furthermore, some advanced approaches add *a presentation layer* (e.g., WebML and OntoWebber) and *a customization layer* (e.g., UWE) to the typical architecture. The presentation layer separates the specification of contents of web pages from that of layouts and presentation styles, thus allowing

different presentation instructions to be easily specified for rendering the same site view. The customization layer separates the specification of customization from other aspects, thus overcoming the problem caused by mixing site specifications with customization specifications, which is forcing web developers to anticipate what can be customized at the time of specifying site views and their presentation instructions. However, there are still a number of limitations, which need to be addressed:

- Relatively little support for the specification of site views (i.e. navigation structures and user interfaces). Although recent approaches like WebML, OntoWebber, and Hera overcome the problem of earlier approaches (e.g., HDM, RMM, OOHDM, HDM-lite, WSDM, and UWE) which is only able to provide methodological support for the specification of site views, only coarse-grained primitives are provided to model typical user interface elements of web pages, such as elements for data presentation, data querying, and data acquisition. No further modelling support is available to allow the adaptation of such typical user interfaces. As a consequence, web developers are only able to express a fixed number of typical user interfaces in terms of the provided primitives. The creation of complex user interface is out of reach at the conceptual level.
- Little support for the specification of the layouts and presentation styles for user interface elements. Although recent approaches like WebML and OntoWebber explicitly separate the user interface elements of web pages from presentation, they do not offer comprehensive support for the specification of presentation instructions. As a consequence, web developers still have to rely on ad hoc approaches, e.g., CSS, and low-level programming, to define and maintain the specification.
- The lack of comprehensive customization support. Although two major customization solutions have been developed in current approaches to address the issue of customization, only limited support is provided. Firstly, as discussed earlier, the compositional-level customization can not scale up, as when the number of user groups or individuals grows, the workload of maintaining a large number of site models becomes too heavy. Secondly, the scope of the derivation-level customization support is limited, as not all aspects of web sites are available for

customization, due to the lack of expressive meta-models for describing the target web site. Furthermore, specific support for the specification of derivation customization requirements is not available. Finally, some approaches, e.g., WebML and Hera, do not separate customization requirements from other aspects of web sites, thus forcing web developers to anticipate what can be customized at the stage of navigation structure design and user interface design.

Chapter 3: Customization design of web applications

In addition to the research field of conceptual web modelling (as described in chapter 2), customization design for web applications has also been addressed in a number of other research areas, including adaptive user interfaces [Good et al., 1984] [Fink et al., 1998] [Langley, 1999], adaptive hypermedia [Brusilovsky, 1996] [Kobsa et al. 2001], information customization systems [Maurino & Fraternali, 2002] [Mostafa, 2002], and ubiquitous computing [Dey & Abowd, 2001] [Kappel et al., 2003] [Landay & Borriello, 2003]. In this chapter, we examine techniques developed on user modelling, user information acquisition, and customization[1]. Furthermore, we investigate tools and approaches which share the goal of supporting the design and development of customized web applications. Our purpose here is to motivate our work on developing a generic customization framework which on the one hand supports customization specification at design time and on the other delivers comprehensive customization support at run time for data-intensive web sites.

3.1 Introduction

As web sites offer information which is potentially interesting to a wide audience [De Bra et al., 2003], the ability to customize content for user groups and individuals is becoming a major requirement for web site design for improving their usability [De Troyer & Leune, 1998], as users with different goals and knowledge may be interested in different pieces of information presented and may use different links for navigation [Brusilovsky, 1996]. Thus, customization should be addressed which

[1] Please note that as mentioned in chapter 2 we do not necessarily distinguish the words customization, personalization, and adaptation in this work, as all of them refer to the specialization of web sites towards user groups or individuals.

specializes a general purpose of web site towards the profiles of user groups or individuals.

Customization has been an issue in computer science for a long time since the end user has been put in the middle of concern when developing interactive applications [Kappel et al., 2003] [Maurino & Fraternali, 2002] [Kobsa et al., 2001] [Fink & Kobsa, 2000] [Brusilovsky, 1996] [Good et al., 1984]. Substantial efforts have been made in developing techniques for customizing software applications towards user groups or individuals.

Firstly, a number of customization techniques regarding user modelling, user information acquisition, and adaptation methods have been developed which aim at i) tailoring the applications' interactive behaviour to knowledge, skills, tasks, and preferences of end users [Langley, 1999] [Fink et al., 1998] [Kobsa et al., 1994] , ii) providing personalized recommendations or personalized assistant [Perkowitz & Etzioni, 2000] [Jenamani et al., 2002] [Menczer et al., 2002], or iii) customizing systems towards the access context indicated by the ubiquity of web applications characterized as *anytime/anywhere/anymedia* [Kappel et al., 2003] [Dey & Abowd, 2001].

Secondly, a number of approaches and tools have been developed which employ these customization techniques to support the design and development of customized web/hypertext-based applications. Examples include i) tools like InterBook [Brusilovsky et al., 1998], TANGOW [Carro et al., 1999], AHA [De Bra & Calvi, 1998], and SETA [Ardissono & Goy, 1999] which come from the area of adaptive hypermedia, ii) approaches like WebML [Ceri et al., 1999b], UWE [Hennicker & Koch, 2000], the extended OOHDM [Schwabe et al., 2002], and Hera [Frasincar et al., 2002] which as described earlier in chapter 2 are developed in the research field of conceptual web modelling, and iii) approaches like the Web Unified Modelling Language (WUML) [Kappel et al., 2001, 2002] which stem from the area of ubiquitous computing.

In the following sections, we will briefly describe the customization techniques and customization tools.

3.2 Customization techniques

3.2.1 User modelling

Four typical user modelling approaches have been developed, which aim to describe information about end users. They include i) the stereotype approach which models a particular user by assigning him or her to one of stereotypes defined in advance, ii) the overlay model approach which focuses on user's knowledge about the domain concepts and defines the user model as a subset of the structural domain model, iii) the goal-based approach which identifies user goals, plans, or tasks in order to provide appropriate information to meet the requirements of individual users, and iv) the context modelling approach which models user's contextual information (e.g., time, location, and device).

The stereotype approach

According to [Kobsa, 1993], the stereotype approach is used to i) identify user groups within the expected user population whose members are very likely to possess certain relevant characteristics, ii) identify a small number of key characteristics which allow one to identify the members of a user group, and iii) represent user groups in an appropriate representation system.

The effectiveness of this approach depends on the quality of the stereotypes, such as the number of different stereotypes known to the system, the accuracy of assignment of users to stereotypes, and the quality of inferences that are drawn from stereotype membership. This approach has been used in a number of systems such as KN-AHS [Kobsa et al., 1994] and AVANTI [Fink et al., 1998].

The stereotype user modelling approach has been used as a basic user modelling technique in a number of customization approaches. For example, user groups are explicitly supported in many conceptual web modelling approaches (e.g., UWE, WebML, OntoWebber, and Hera), which classify users according to their roles involved in target web sites. In the area of information customization systems, rule-based user modelling approaches are developed based upon the stereotype approach,

which allow web administrators to define user groups and specify rules based on user demographics, information collected during registration or other statistics in advance. Rules are used to reason and create particular content to particular users. The most typical example may be Broadvision[2]. This kind of user modelling is usually deployed in the situation when there are limited documents, products or services the system provide, and the human web administrator can write a small number of rules manually for tailoring the interfaces for users.

The overlay model approach

The overlay model approach defines the user model as a subset of the structural domain model. The domain model is represented as a network with concepts and links. Concepts represent different kinds of knowledge elements or objects, and the links represent relationships between concepts. For each concept, a user model stores some value to represent the user knowledge level of this concept. The measure of the value can be Boolean, a qualitative measure or a quantitative one.

To explore this approach, developers need to construct user models consisting of a set of variables. These variables are used to represent the knowledge of a user by means of concepts. These user models are typically updated during the process of user interaction with systems. This user modelling approach has been employed in systems like ELM-ART [Brusilovsky et al., 1996], InterBook, AHA, and KBS-hyperbook [Henze and Nejdl, 1999].

The goal-based approach

The goal-based approach identifies user goals, plans, or tasks for providing appropriate information to meet the requirements of individual users. The user model includes a set of possible user goals, tasks or plans. PUSH [Hook, 1996] and TANGOW are typical goal-based examples.

---

[2] http://www.broadvision.com (Accessed 18 October 2004)

The context modelling approach

The context modelling approach focuses on capturing features (e.g., time, location, and device) of user's contextual information. It is typically used in ubiquitous computing systems for enabling customization according to the access context of an end user. This modelling approach has been used in WUML.

3.2.2 User information acquisition

A number of user information acquisition techniques have been developed, which enable the adaptive or customization support. Typical examples include explicit user input, acquisition rules, plan recognition, stereotype reasoning, collaborative filtering, web usage mining, and context information computing.

- Explicit user input typically takes place in an initial phase of system usage as a format of questionnaire or a form provided to ask users to respond. A number of adaptive systems (e.g., KN-AHS, AVANTI, and AHA) employ this approach to obtain initial information about users.
- A number of systems explore acquisition rules to acquire user information during the interaction process. In most cases, acquisition rules refer to observed user actions or a more or less straightforward interpretation of user behaviour. The acquisition rules can be domain-independent and domain-dependent. KNOME [Chin, 1989] and BGP-MS [Kobsa et al., 1994] are generic user modelling systems which explore domain-independent acquisition rules to obtain user information dynamically.
- A typical plan recognition system consists of a plan library served as a task knowledge base, and of a reasoning mechanism to identify the current goal or task from the observed interactions. This approach is suitable for systems with a small number of possible goals and a small number of ways to achieve these goals.
- The stereotype reasoning is used to reason about which stereotype the current user belongs to by evaluating the conditions of available stereotypes with the observed user actions. This approach is mainly used in systems explores stereotype modelling and overlay user modelling.

- The collaborative filtering approach is developed in the area of electronic-commerce systems, which acquires user information from like-minded users. This approach takes explicitly user information in the form of ratings and preferences and via the collaboration of like-minded users, and makes interface adaptations. RINGO [Shardanand & Maes, 1995] is a typical example to recommend items a person might enjoy by making predictions about items based on feedback from many different users. This method has been used by a number of vendors on the World Wide Web, including amazon.com[3], to sell books and other items.
- Web usage mining primarily uses data mining algorithms to automatically discover and extract patterns from web usage data and predict user behaviour while users interact with web applications. The adaptive web sites [Perkowitz & Etzioni, 2000] is a classic example of using web usage mining to improve the presentation and organization of web sites by learning from visitor access patterns. The main advantage of this methodology is that it can reduce the need for obtaining subjective user ratings or registration-based personal preferences.
- Context information acquisition understands and acquires the context information of the end user when he or she accesses web applications. WUML is an example which uses this kind of technologies to capture user contextual information.

3.2.3 Adaptation methods

Adaptation in customization systems typically takes place at two levels: *link-level* and *content-level* [Brusilovsky, 1996] [Rossi et al., 2001]. Link-level adaptation adapts the navigation structure towards end users. Examples of link-level adaptation include link sorting which ranks all the links, link hiding which removes irrelevant links from web pages, and link annotation which associates additional information with the hyperlinks. Content-level adaptation adapts the content of web pages towards the needs of end users. Examples of content-level adaptation include:

---

[3] http://www.amazonl.com (Accessed at 16 June 2005)

- The *conditional* text approach which divides a concept into several chunks of text. Each chunk is associated with a condition on the level of user knowledge represented in the user model. As a result, the system presents only the chunks where the condition is true. ITEM/IP [Brusilovsky, 1992] and C-book [Kay & Kummerfeld, 1994] follow this approach.
- The *Stretchtext* approach which defines additional text as explanations of the potential hot words. Clicking on hot-word results in additional text being displayed in a pop-up window. MetaDoc [Boyle & Encarnacion, 1994], Anatom-Tutor [Beaumont, 1994], and KN-AHS use this approach for content adaptation.
- The *page variant* approach which keeps two or more variants of a same page with different presentations. Each variant is prepared for one or more of the possible user groups. The customization mechanism is realized by selecting the page variant according to the user stereotype. Anatom-Tutor [Beaumont, 1994] with background stereotypes, and KBS-Hyperbook with knowledge-level stereotypes are classic example of page variant adaptation. Composition-level customization developed in conceptual web modelling approaches uses this approach to assign different site view of the specified page node for different user groups.
- The *fragment variant* approach which annotates web pages with a number of variants of explanations. Each variant explanation is designed for one particular user group. Customization is realized by selecting the appropriate variant for content presentation according to the group that an end user belongs to. KN-AHS, AHA [De Bra & Calvi, 1998], AVANTI, and ADAPTS [Brusilovsky & Cooper, 1999] have used this approach for content adaptation.
- The *frame-based* approach which represents information about a concept in the form of frame. Slots of a frame can contain several explanation variants of the concepts. Special presentation rules are used to decide which slots should be presented and in which order. This approach has been used in Hypadapter [Hohl et al., 1996] and in those conceptual web modelling approaches which employ rules to compute the content of particular information fragments.

Please note that it is often that the link-level adaptation and content-level adaptation are combined together to provide customization support. For example, variants can

be attached to links, which rely on rules to compute the visibility or the URL of the linked web page.

3.3 Customization approaches

In this section, we examine approaches developed in a number of research directions which support the design and development of customized web (or hypertext-based) applications. The purpose is to outline the problems with the current customization approaches and motivate our work on customization modelling.

3.3.1 The authoring tools for adaptive hypermedia systems

The basic components of adaptive hypermedia systems usually include an application domain model, a user model, a set of adaptive methods, and an adaptive engine [Brusilovsky, 1996] [Brusilovsky, 2001] [De Bra et al, 1999a]. The application domain model describes how the information is structured and linked together. The user model describes user profiles (e.g., characteristics, goals, preferences). The adaptive methods describe how to perform adaptation towards user profiles. The adaptive engine produces personalized views for user individuals according to the knowledge, plans, and goals of user individuals.

Researchers in this area have identified that building adaptive hypermedia applications is a time consuming task and developed a number of authoring tools to simplify the design and development process. Examples include InterBook, TANGOW, AHA, and SETA:

- InterBook is a tool for authoring and delivering adaptive electronic textbooks on the World Wide Web. It relies on the individual order of learning goals and adaptive guidance to build personalized adaptive courses from the same course material. The major components include a domain model which comprises a set of domain concepts (i.e. elementary pieces of knowledge) for the specified domain, a student model which maintains an estimation of the student knowledge level for each domain concept, a model of student's learning goals which represents a set of

concepts to be learned, and a glossary which resembles the pedagogical structure of the domain knowledge. Authoring electronic text books with InterBook involves in manually structuring and annotating text content. The InterBook Server generates adaptive content on the fly according to the profile of students.

- TANGOW supports the design and development of adaptive web-based coursers. It models web courses by means of teaching tasks and rules. A teaching task is the basic unit that appears in the learning process, and is modelled by means of a set of attributes (e.g., *name*, *content*, *composition type*, and *ending requirements*). A rule defines how a teaching task is divided into subtasks, and specifies the requirements for a teaching task to be applicable. The TANGOW approach relies on teaching tasks, rules, student profiles and actions to guide the student towards the desired goals during the learning process by deciding the next set of achievable tasks step by step. The adaptive feature of the target learning environments is implemented from three dimensions: task decomposition can be different depending on the fact stored in the student profile, rules are defined with preconditions related to other tasks achievement, and different multimedia elements are used to generate a web page depending on the student profile.
- AHA supports the adding of adaptive features to web sites by allowing building a domain model which describes the problem domain, creating a user model which expresses the user knowledge level of concepts contained in the domain model, and annotating web pages using the provided method.
- SETA supports the construction of adaptive web stores. It exploits taxonomies to represent products (with roots describing general product categories and subclasses expressing more specific products) and uses stereotypical information to handle user models. Customization is achieved by applying the specified presentation rules and assembling web pages on the fly.

In summary, these tools provide comprehensive customization support for their target hypermedia applications. They make use of complex user modelling techniques to describe information about end users at design time and powerful adaptive engines to deliver customized applications at run time. They however target specific domain. For example, tools like InterBook, KBS-Hyperbook, and

TANGOW support the generation of electronic books and web courses. AHA facilitates the building of learning systems. SETA focuses on web store generation. Another limitation is that they do not provide high level support for the authoring of the target applications. Specifically, no meta-models are provided, which support the i) specification of web pages and customization requirements and ii) the association of customization requirements with web pages at the conceptual level. Web pages are either created manually by developers on their own by means of ad-hoc approaches (e.g., in AHA), or generated automatically (e.g., in InterBook, TANGOW and SETA).

3.3.2 Conceptual modelling approaches

As described in chapter 2, a number of conceptual web modelling approaches consider customization issues as an important modelling dimension and provide support for customization design. Examples include WebML, OOHDM, UWE, and Hera. They support one-to-one web site delivery by providing a personalization model or a customization model that describe user groups, individual users, customization context, and customization rules. The customization frameworks of these proposals offer mechanisms that allow site designers to pre-define customization rules or different site views to enable delivering customized web applications for user groups or individual users.

These approaches have already been described in chapter 2. In this section, we present the Web Unified Modelling Language (WUML) which dedicates itself to customization modelling of web applications and thus has not been described in chapter 2.

The Web Unified Modelling Language (WUML)

WUML proposes a generic customization model which allows personalizing web applications towards the context implied by ubiquity that can be characterized as *anytime/anywhere/anymedia* paradigm. The customization model comprises two components. One is the context model which describes detailed information about

the environment of a web application and the web application itself. The other component is customization rules which describe customization requirements, e.g., when and how to perform customization actions.

The context model is divided into a physical context model and a logical context model. The physical context model represents the level of environmental sensors at a very low level. For example, the context of *location* captures information about from which the web application is accessed. The logical context model represents logical information in terms of profiles for each component of the physical context model. For example, the location profile enriches the semantics of the physical location context.

Customization rules are specified in terms of *event/condition/action* in WUML. The event part describes the events which are able to trigger the corresponding rule. The condition part is evaluated when the rule is triggered by an appropriate event. The action part describes certain adaptations of the application that should take place when the corresponding condition is satisfied.

The WUML approach does not support the modelling of other aspects of web applications, including domain and site views. As a consequence, it requires the web applications to be *sliced* into a stable part which comprises context-independent structures and a variable context-dependent part which describes the subjects of the adaptations. Furthermore, to realize customization, web developers need to define adaptation hooks which represent adaptation actions in web applications.

3.3.3 Other approaches

MyYahoo

The MyYahoo approach [Manber et al., 2000] is a typical example of user specified customization methodology, which facilitatesusers to select preferred modules from hundreds of those available. This approach provides comprehensive customization support for end users only when they are willing to specify their customization requirements (such as modules, page presentation styles and layouts templates). As

studied in [Manber et al., 2000], many users do not bother to manually customize their web pages. Thus, techniques such as automatic user information acquisition should be employed to support dynamic customization (i.e., customization according to the contextual information of end users).

Personalized Information Description Language (PIDL)

The Personalized Information Description Language (PIDL) [W3C, 1999b] is an approach, which provides a framework for applications to progressively process original contents and append personalized versions in a compact format. The key idea of PIDL is to encapsulate both the original contents and the processed personalization in a single XML document. It slices the original content into blocks, and adds user specified personalization methods to enable produce personalized content. While PIDL provides a simple but powerful solution to customizing pre-defined documents, it cannot handle dynamic documents whose content is not available before run time.

Customization information systems

Recently, a number of customization information systems have been developed to generate recommendations for end users such as the customized index synthesis (Jenamani et al., 2002; Perkowitz & Etzioni, 2000) and adaptive online assistants [Menczer et al., 2002]. These systems typically employ the web usage mining technology [Mulvenna et al., 2000] to acquire information about end users and thus produce useful recommendations. While they employ a number of customization techniques for providing personalized help for end users, they do not customize their target web applications.

3.4 Conclusions

In this chapter we have examined current techniques on user modelling, user information acquisition, and customization methods. These techniques can be re-used in our work. Thus we can focus on developing a comprehensive customization

framework which supports the specification of customization requirements at design time and for the delivery of customization at run time. Specifically, the stereotype user modelling approach can be used to define user groups to classify users in the target web site. The overlay user modelling approach can be employed to capture users' knowledge about domain concepts. The context modelling approach provides basis for capturing contextual information of end users. The user information acquisition techniques can be used to support the acquisition of user profiles. Finally, the customization methods facilitate the derivation of customized views for users.

In addition to the customization techniques, we have also briefly described tools and approaches which support customization design for web applications. In particular, we have investigated a number of authoring tools which support the authoring and generation of adaptive hypermedia applications. Although as described earlier, they lack high level support for the specification, these tools provide comprehensive support for user modelling and the generation of adaptive applications. In particular, the customization methods developed in these tools can be re-used in our work to provide comprehensive support for the derivation of customized views.

We have also examined a customization modelling approach, WUML, which dedicates itself to customization design. It provides a complex context model to capture users' contextual information and employs customization rules to enable the derivation of customized views for users. WUML provides some valuable principles and lessons for us to build our customization framework. Firstly, contextual information of end users can be used to drive dynamic customization support by means of applying rules and user contextual information to reason upon site specifications. Secondly, to provide appropriate support for the design and development of web sites, an approach should support the modelling of customization as well as other aspects. Otherwise, like WUML, the approach would require the web applications to be *sliced* into a stable part which comprises context-independent structures and a variable context-dependent part which describes the subjects of the adaptations. Furthermore, to realize customization, adaptation hooks need to be defined which represent adaptation actions.

Chapter 4: An overview of the OntoWeaver design framework

In this chapter we present an overview of our approach OntoWeaver to the design and development of data-intensive web sites. We begin by re-stating the limitations of current high level design approaches described in chapter 2. We then clarify the requirements placed upon high level design approaches for addressing these issues. Thereafter, we present an overview of our approach with respect to the research goals, assumptions, restrictions, and major contributions.

4.1 Limitations of current web modelling approaches

Data-intensive web sites are web applications which provide dynamic access to large sets of data typically stored in underlying databases. Building such web sites is a complex task [Fernandez et al., 1997], which involves not only technical issues, but also organizational, managerial and artistic issues [Morville & Rosenfeld, 1998] [Fraternali, 1999] [Koch, 1999]. Furthermore, the role of end users needs to be augmented into the design [Brusilovsky, 1996] [De Troyer & Leune, 1998] [Gómez et al., 2000] [Kappel et al., 2000] [Kobsa et al, 2001]. This makes the situation more difficult, as comprehensive customization support needs to be provided, which allows the target web site to be responsive to the individual needs of end users. To address these problems, a number of approaches have been proposed, which aim to simplify the design and development of web sites by allowing them to be specified at a high level of abstraction and by being able to automatically or semi-automatically produce implementations from high level specifications [Fraternali, 1999] [Koch, 1999] [Retschitzegger & Schwinger, 2000].

As studied in chapter 2, these approaches typically distinguish three layers to describe data-intensive web sites: *a data layer*, which describes the underlying

domain data of the target web site, *a navigation structure layer*, which describes the navigation structures of the target web site, and *a user interface layer*, which describes the user interfaces of the target web site. At the same time, they provide models to address each layer accordingly. Firstly, these approaches provide comprehensive support to facilitate the design of navigation structures. Secondly, most approaches address the design of user interfaces. Finally, most approaches take customization into consideration and provide customization support for the target web site to a variety of levels.

Some advanced approaches add *a presentation layer* (e.g., WebML and OntoWebber) and *a customization layer* (e.g., UWE and WUML[1]) to this typical architecture. The presentation layer separates the specification of contents of web pages from that of layouts and presentation styles, thus allowing different presentation instructions to be easily specified for rendering the same site view. The customization layer separates the specification of customization from other aspects, thus overcoming the problem caused by mixing site specifications with customization specifications, which is forcing web developers to anticipate what can be customized at the stage of specifying site views. However, as clarified in chapter 2, there are still a number of limitations, which need to be addressed:

- Relatively little support for the specification of site views (i.e. navigation structures and user interfaces). Earlier approaches (e.g., HDM, RMM, OOHDM, HDM-lite, WSDM, and UWE) are only able to provide methodological support for the specification due to the lack of explicit formal meta-models. Although recent approaches like WebML, OntoWebber, and Hera have overcome this problem, only coarse-grained primitives are provided to model typical user interface elements of web pages, such as elements for data presentation, data querying, and

[1] As pointed out in chapter 2, WUML does not completely separate the specification of customization requirements from other aspects of web sites, as it requires adaptation hooks to be defined in the specifications of web sites.

data acquisition. No further modelling support is available to allow the adaptation of such typical user interfaces.

- Little support for the specification of layouts and presentation styles for user interface elements. Although recent approaches like WebML and OntoWebber explicitly separate the user interface elements of web pages from presentation, they do not offer comprehensive support for the specification. Web developers still have to rely on ad hoc approaches to define and maintain the specification. This is time consuming in particular when different presentations are required for rendering the same site view for different purposes.
- The lack of comprehensive customization support. As described in chapter 2, two solutions have been developed in current approaches to support customization. They are the composition-level customization, which allows the construction of different site views at design time, and derivation-level customization, which allows the derivation of different site views at run time. However, they can only provide limited customization support. Firstly, the compositional-level customization can not scale up, as when the number of user groups or individuals grows, the workload of maintaining a large number of site models becomes too heavy. Secondly, the scope of the derivation-level customization support is limited, as not all aspects of web sites are available for customization, due to the lack of expressive meta-models for describing the target web site. Finally, specific support for the specification of derivation customization requirements is not available.

4.2 A solution

To address the issues described above, except adding a presentation layer and a customization layer to the typical abstract architecture of data-intensive web sites as most recent approaches do, two major components should be provided.

One is a comprehensive set of meta-models, which captures features of all aspects of site views, including navigation structures, user interfaces of web pages, and their presentation styles and layouts, thus allowing the target web site to be specified in a declarative and re-usable format at the conceptual level. The nature of declarative

specification of the entire target web site offers many potential benefits over traditional hard-coded specification and partially hard-coded specification. Firstly, it facilitates the construction of tools to assist developers at design-time and web site administrators and end-users at run-time, for the declarative model provides a common representation which can be reasoned about. Secondly, it supports rapid prototyping and iterative development. Developers can construct prototype systems rapidly using conceptual level specification. Finally and most importantly, it opens up a number of possibilities with respect to intelligent analysis and management. Customization and site design critiquing are such examples. Specifically, as the entire site model is available for reasoning, the scope of customization is not limited. Moreover, recommendations can be produced for web developers to improve their design by applying a set of critiquing rules which are able to reason upon the entire site model.

The other component is a customization framework, which is able to exploit the advantage of the declarative specification of web sites and offers comprehensive customization support at design and run time. In particular, the customization framework should separate the specification of customization requirements from other aspects of web sites. Thus, web developers do not have to anticipate what can be customized at the phase of specifying other aspects of web sites. Furthermore, it should provide high level support for the specification of customization requirements. Hence, web developers do not have to rely on ad-hoc approaches to augment individual's roles into site specifications as they have to in current approaches. Finally, it should provide a customization engine which performs inferences on site specifications and derives customized views for individual users at run time.

4.3 The OntoWeaver approach

One major goal of this research is to produce an approach which is able to provide components as clarified above, thus addressing the limitations of current approaches. In particular, in order to allow meta-models to be specified in a format that provides

shared semantics which allows knowledge sharing and exchanging in the life cycle of web sites, we choose using ontologies [Gruber, 1993] to represent meta-models and their instantiations i.e. specifications of web sites.

An ontology is an explicit formal specification of a conceptualization [Gruber, 1993]. It supports the formal specification of concepts and relations between them that can be shared in a certain domain [Neches et al., 1991] [Borst et al., 1995]. By using ontology, web site specification is formalized and can be shared during the life cycle of a web site, thus easing the job of web site maintenance and management, as all components are declarative, re-usable, and represented in shared semantics. Furthermore, as the provided explicit shared semantics, the target web sites can be picked up by semantic-aware agents and thus benefit from intelligent services (e.g. indexing, searching, and customization) provided by third parties.

The use of ontologies to drive the generation of software tools has been demonstrated in the research area of knowledge acquisition and semantic web portals, where a number of tools have been developed which use domain ontologies to drive the generation of knowledge acquisition tools [Eriksson et al., 1994] [Grosso et al., 1999] [Motta et al., 2000] and semantic web portals [Corcho et al., 2003] [Volz et al., 2003]. This research goes one step further. It applies ontologies to model the target software which is data-intensive web sites, thus facilitating the design and development process. In particular, as clarified above, our goal is to applying ontologies to address the limitations associated with current web modelling approaches.

This research has produced both an approach called OntoWeaver and a tool infrastructure called OntoWeaver tool suite. The OntoWeaver approach provides a comprehensive set of ontologies to model data-intensive web sites, including *a site view ontology* and *a presentation ontology*. The site view ontology addresses the first limitation discussed earlier by providing fine-grained modelling support for user interfaces and navigation structures of data-intensive web sites. The presentation ontology captures the features of the layouts and presentation styles of user interface elements. It addresses the second limitation of current approaches.

The provision of expressive meta-models partially addresses the third limitation of current approaches, as all aspects of general purposes of data-intensive web sites are represented declaratively and are available for customization. Thus, the scope of customization is not limited. To exploit this advantage and to provide comprehensive customization support, the OntoWeaver approach provides *a customization framework*, which i) relies on rules to specify customization requirements of individual users, ii) provides *a customization rule model* to support the specification at design time, and iii) provides a customization engine to deliver customization support at run time.

4.3.1 An overview

Figure 4.1 shows an overview of the OntoWeaver approach. It distinguishes four layers to abstract data-intensive web sites, which include *a data layer*, *a site view layer*, *a presentation layer*, and *a customization layer*. Please note that this architecture is a logical one not a physical one (in which how data stores).

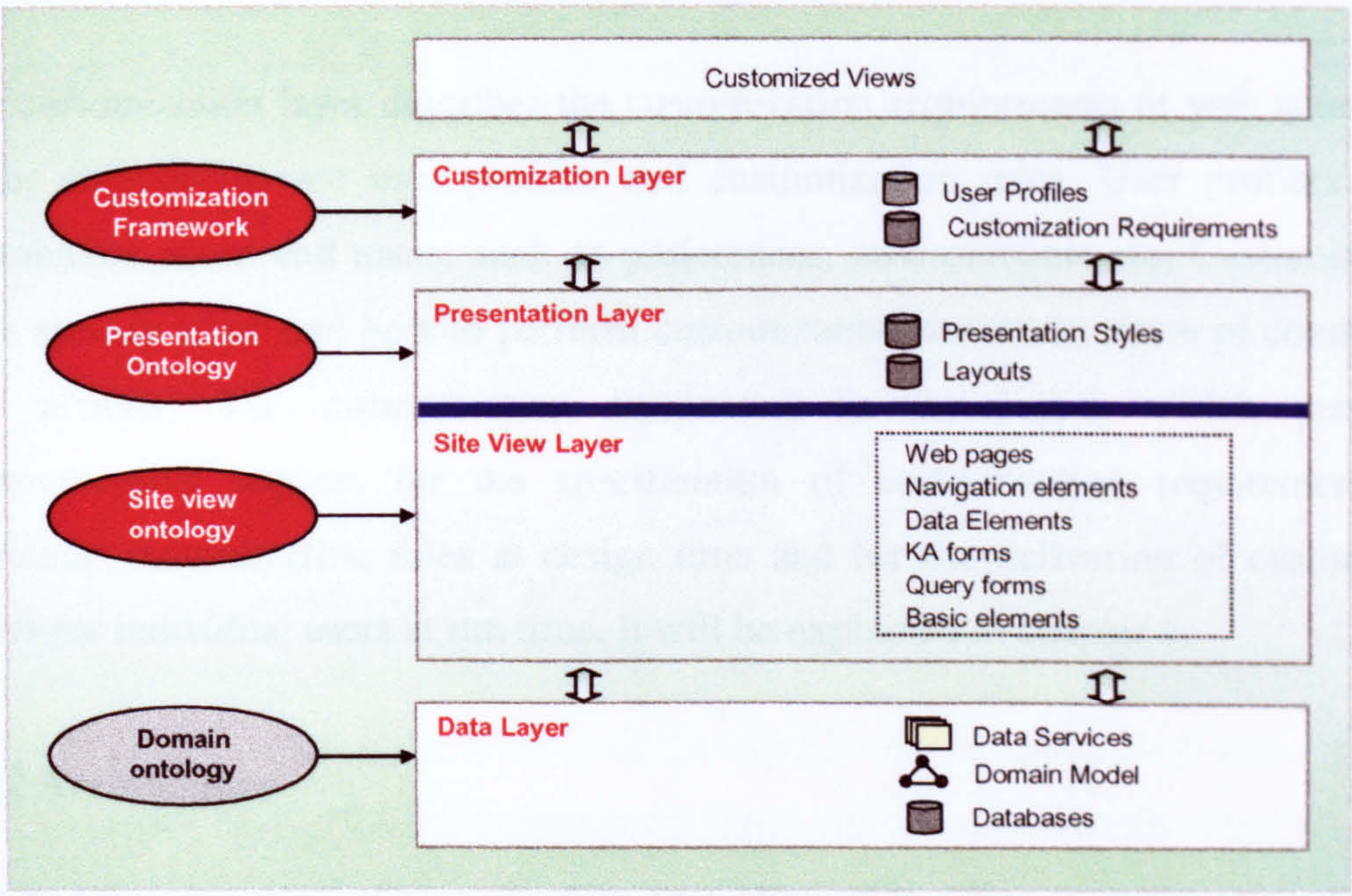


Figure 4.1 An overview of the OntoWeaver approach

Like in the typical architecture of current approaches described earlier, the data layer describes underlying domain data models and data objects. OntoWeaver relies on domain ontologies to specify this layer.

The site view layer is the combination of the navigation structure layer and the user interface layer. The combination avoids the overlapping between navigation elements and user interface elements. For example, web pages can be seen as navigation elements as they are navigation nodes in navigation structures, as well as user interface elements as they are compositions of user interface elements. In order to provide appropriate support for the specification of this layer, an expressive meta-model is crucial in web site design. The site view ontology is developed according to this requirement.

The presentation layer describes layouts and presentation styles for site view elements. To allow presentation instructions to be specified at a high level without having to commit to implementation issues, the presentation ontology is developed, which captures features of this layer. Both the site view ontology and the presentation ontology will be clarified in chapter 5.

The customization layer describes the customization requirements of web sites. The major components are user profiles and customization rules. User profiles store information about end users, such as preferences, environments etc. Customization rules specify when and how to perform customization actions in terms of conditions and actions. The customization framework is developed, which provides comprehensive support for the specification of customization requirements in particular customization rules at design time and for the delivering of customized views for individual users at run time. It will be explained in chapter 6.

4.3.2 Architecture

Figure 4.2 shows the architecture of the OntoWeaver framework. It accepts a domain ontology as input and produces a customized data-intensive web site for individual users. A typical design process in OntoWeaver proceeds by iterating the following

steps: i) designing the domain ontology; ii) specifying navigation structures and composing user interfaces; iii) defining layouts and presentation styles, and iv) expressing customization requirements.

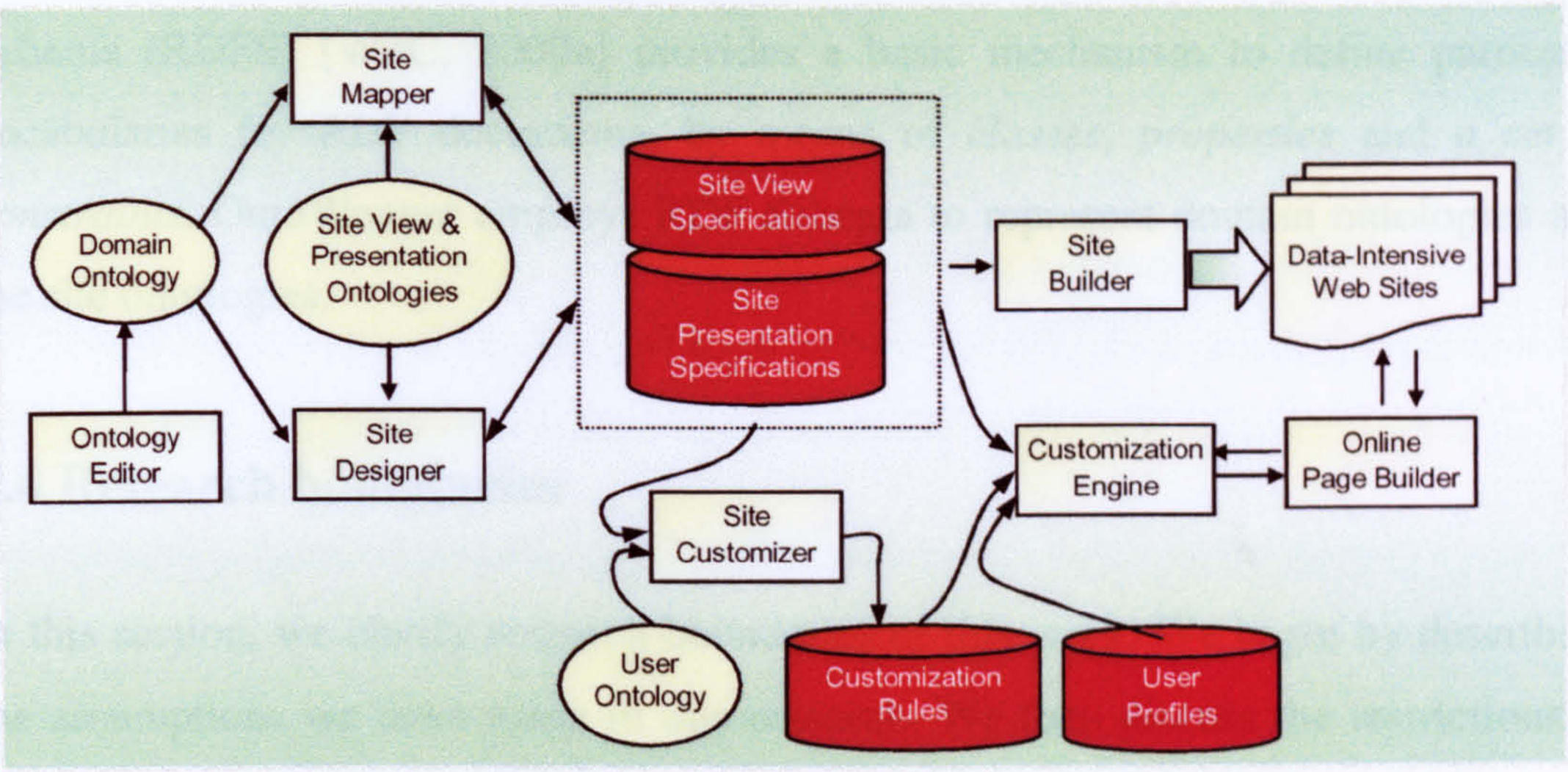


Figure 4.2 The architecture of the OntoWeaver framework

As shown in figure 4.2, OntoWeaver provides a set of tools to support the design activities and the generation of customized data-intensive web sites. Specifically, the *Ontology Editor* allows developers to create and edit the ontologies associated with the target web site. The *Site Designer* supports the design tasks needed for specifying a data-intensive web site. The *Site Mapper* produces a default specification for the target web site and is responsible for re-engineering the web site specification after the domain ontology has been modified. As shown in the figure, tools Site Designer and Site Mapper accept the specified domain ontology and the site view ontology as input and produce declarative specifications describing site views and their presentation instructions for the target web site. The *Site Builder* validates the site specifications and compiles site specifications into web site implementations. The *Site Customizer* supports the activity of specifying customization requirements. The *Customization Engine* performs inferences upon the site specifications and produces customized site specifications. The *Online Page Builder* generates customized web pages on the fly from the customized site specifications. The OntoWeaver tool suite will be described in detail in chapter 7.

The OntoWeaver approach uses a semantic web standard, the Resource Description Framework (RDF) [W3C, 1999], to represent all aspects of web sites. RDF provides a foundation for processing metadata, which enables interoperability between applications that exchange machine-understandable information on the Web. RDF Schema (RDFS) [W3C, 2000a] provides a basic mechanism to define particular vocabularies for RDF documents, by means of *classes*, *properties* and *a set of constraints*. OntoWeaver employs RDF Schema to represent domain ontologies and the site ontologies.

4.4 Research boundaries

In this section, we clarify research boundaries of this work. We begin by describing the assumptions we have made in this research. We then discuss the restrictions of our approach and potential solutions to these restrictions.

4.4.1 Work assumptions

One major assumption we have made in this research is that the underlying heterogeneous data sources of a web site are integrated together in an agreed global schema by technologies such as mediated integration [Wiederhold, 1992] or data warehousing [Hammer et al., 1995]. While mediated data integration approaches perform integration at run time by e.g. querying data sources by means of the provided global schema, data warehousing approaches integrate data in advance of queries.

As will be clarified later in chapter 5, the OntoWeaver approach relies on the conceptual links between site views and the underlying domain data model (i.e. the agreed global schema) to realize dynamic access to domain data for web sites. Hence, as long as a global schema is provided, the OntoWeaver approach can provide comprehensive support for the design and development, as the target web site presents data sources in different representations by means of the global schema. However, in the case of using mediated approaches for data integration, minimal programming efforts are required to set up the connection between the OntoWeaver

data services (e.g. data retrieving, querying, and acquisition) with the mediation (e.g. querying) functions developed in data integration technologies, thus allowing the integration functionality taking place at run time. On the other hand, when using data warehousing approaches for integrating data, no such requirements are required.

Another assumption we have made is that the domain data models are represented in terms of ontologies and in particular in RDF Schema. Thus, each entity defined in a domain model can be referenced by means of its Universal Resource Identifier (URI). As a result, dynamic access to underlying domain data can be specified at the conceptual level by means of defining conceptual links in site view components.

The third assumption we have made in this research is that user profiles are gathered by external tools which are specialized for user information acquisition and they are represented in terms of the pre-defined user ontology. Thus, our approach, OntoWeaver, focuses on the specification of customization requirements at design time and the support of customization at run time.

4.4.2 Restrictions of the OntoWeaver approach

One restriction of this research is that the functionalities of the target data-intensive web sites of OntoWeaver are restricted to generic functionalities, such as data presentation, data acquisition, and data querying. Due to the fact that the approach is not specialized in the design and development of functionalities or services, domain specific functionalities can only be achieved in an expensive way by means of programming. This can be solved by re-using web services developed by third parties, which allow the sharing of functionalities across the Internet. In this context, an OntoWeaver extension called OntoWeaver-S is built in this research, which provides high level support for the integration of web services into data-intensive web sites. This component will be explained in chapter 7.

Another restriction of this research is that the OntoWeaver extension OntoWeaver-S is web service platform dependent. This is mainly because i) there was no standard language available at the time when we carried out this research, which supports high

level specification of web services, and ii) there was only one fully implemented web service platform, IRS-II [Motta et al., 2003], which allows us to test our idea of integrating web services into data-intensive web sites. Hence, we chose IRS-II as the web service platform to support the specification, discovery, and the invocation of web services. Nevertheless, the limitation caused by this restriction will be addressed in future research by means of integrating other web service languages into OntoWeaver-S in terms of the IRS-II web service specification framework.

4.5 Major contributions

This research produces OntoWeaver, which uses ontologies to drive the design and development of data-intensive web sites. OntoWeaver overcomes the problems of current approaches by providing *a site view ontology*, *a presentation ontology*, and *a customization framework*. Specifically, the site view ontology provides fine-grained modelling support for the creation of complex user interfaces and navigation structures. The presentation ontology captures the features of layouts and presentation styles of user interface elements. These two explicit meta-models allow the target web site to be represented in a declarative and re-usable format, thus enabling high level support for design, maintenance, and customization. The customization framework exploits this advantage and provides comprehensive customization support for the target web site at design as well as run time. Furthermore, a tool suite is produced in this research, which supports the design and development of data-intensive web sites upon the basis of the OntoWeaver approach. In addition, as will be described in chapter 7, the tool suite also provides facilities for site design critiquing, maintenance, and customization. Finally, OntoWeaver-S is produced, which provides high level support for the integration of web services into data-intensive web sites. This component will be explained in chapter 8.

4.6 Summing up

While current conceptual web modelling approaches do provide comprehensive methodological support for the design of data-intensive web sites, they suffer from a

number of limitations, e.g., the lack of appropriate support for the creation of complex user interfaces, for the specification of layouts and presentation styles, and for customization. The aim of this research is to apply the notion of ontology to model all aspects of data-intensive web sites and address these limitations.

As mentioned earlier, we have made several assumptions in order to put our focus on the limitations associated with current web modelling approaches. These assumptions include i) the underlying domain data of target web sites have been integrated together in terms of the specified domain ontology, ii) domain data and their structures are represented by means of RDF and RDF Schema, thus all entities can be reached from outside, and iii) user profiles are gathered by means of external tools.

There are two limitations associated with this research. One is that the functionalities of the target data-intensive web sites of this work are limited to the access and manipulation of the underlying domain data objects. This limitation has been partially addressed by the work on OntoWeaver-S which supports the integration of web services with data-intensive web sites. The other limitation of this research is that the web service integration that OntoWeaver-S supports is IRS-II dependent. How to develop a more generic method for web service integration will be our future work.

The major contributions of this research include the OntoWeaver site ontologies which support the specification of all aspects of data-intensive web sites, the customization framework which provides customization support, the OntoWeaver tool suite which facilitates the design and development web sites, and OntoWeaver-S which supports the integration of web services into web sites.

Chapter 5: Modelling data-intensive web sites

As clarified in chapter 4, the OntoWeaver solution to the limitations of current web modelling approaches is providing a comprehensive set of site ontologies and a customization framework to capture features of data-intensive web sites. In this chapter we focus on the site ontologies. We begin by describing a data-intensive web site scenario which will be used for illustrating the site ontologies throughout the chapter. We then describe one component of the site ontologies - the site view ontology, which characterizes navigational structures and user interfaces of data-intensive web sites. In particular, we explain why the site view ontology can provide comprehensive support for the specification of complex site views. Thereafter, we illustrate the other component of the site ontologies – the presentation ontology, which abstracts the look and feel of site views. Next, we describe the generation of web site implementations from ontologies. Finally, we describe related work and restate the major contributions of the site ontologies.

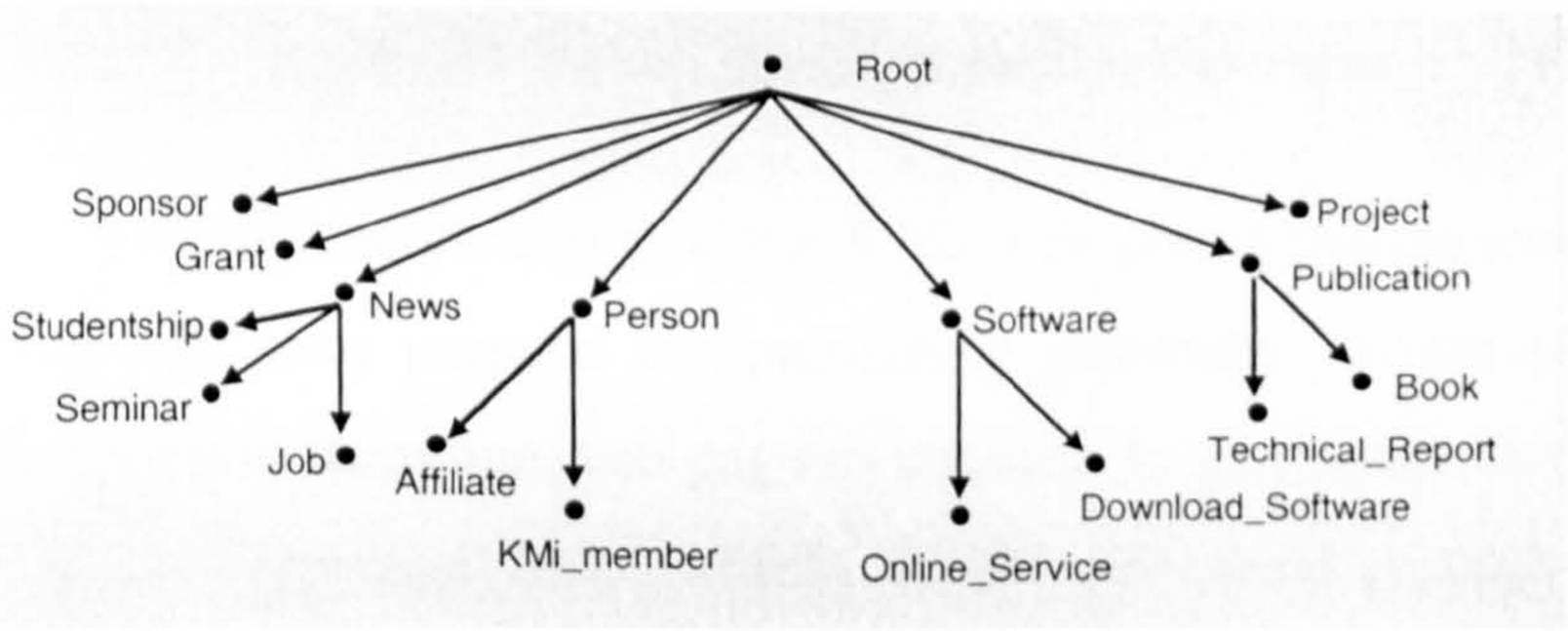


Figure 5.1 The main classes of the domain ontology of the KMi web portal

5.1 A data-intensive web site example

For the purpose of illustrating the OntoWeaver approach to modelling data-intensive web sites, we specify a new web portal for the Knowledge Media Institute (KMi) at the UK's Open University. This web portal presents information about the KMi organization, e.g., *news*, *events*, *research projects*, *publications*, and *persons*, which

is formalized in terms of the KMi domain ontology. Figure 5.1 shows the main classes of the domain ontology. The web portal allows general users to browse and query the underlying domain data and allows advanced users to add new data entries.

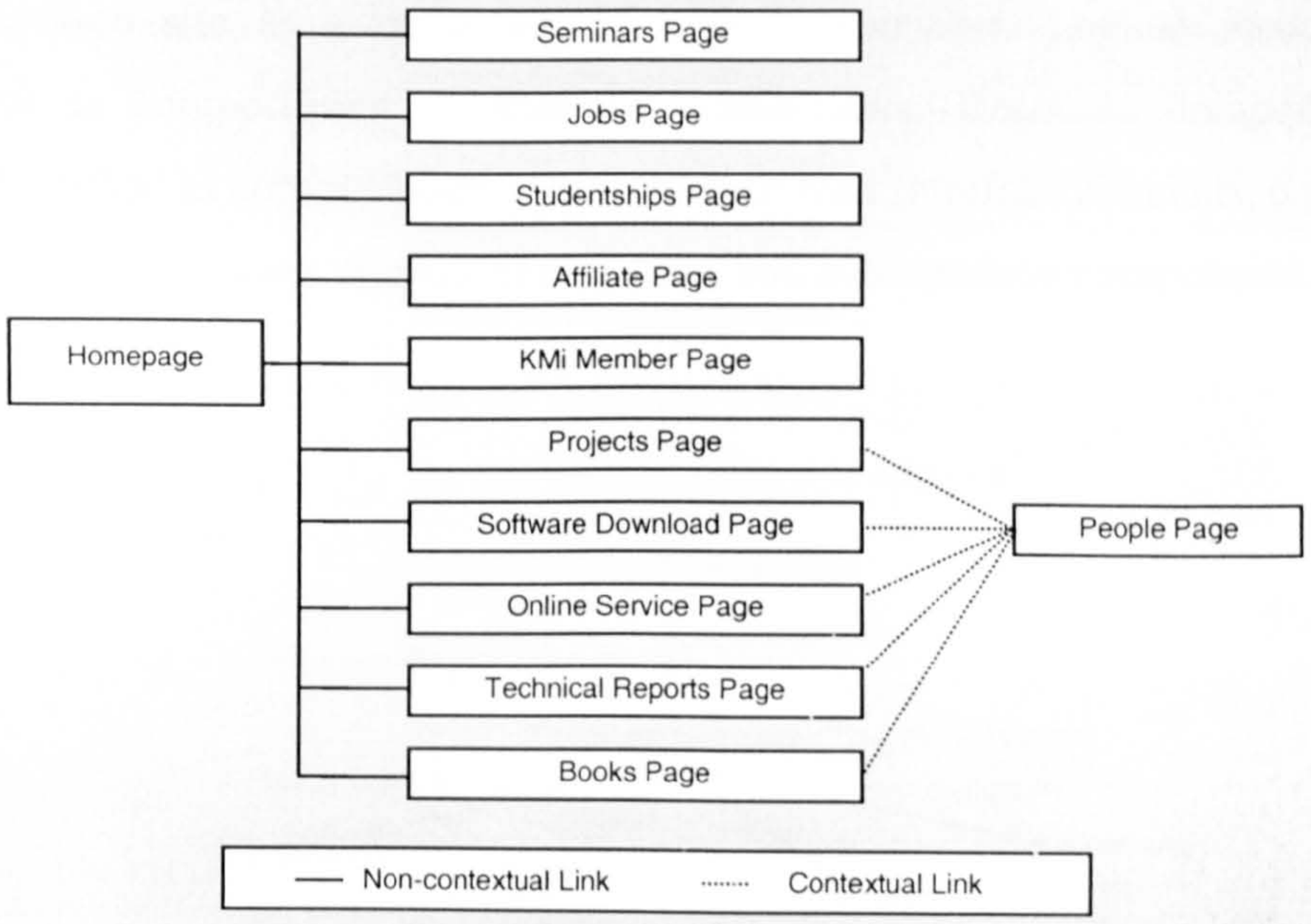


Figure 5.2 The site structure of the KMi Web Portal

Figure 5.2 shows the site structure of the KMi web portal (for general users). Each web page has its own purpose for publishing particular information; each link facilitates navigation from one web page to another. In particular, there are two link patterns. One is *contextual links*, which carries contextual information from the source web page to the destination web page. For example, the link between *the project web page* and *the people web page* is contextual, as it allows navigation from the brief information about project members to detailed information. The other is *non-contextual links*, which does not require any information to be carried on when navigating from the source to the target web page. In the following sections, we will use the KMi web portal as the test case to clarify the modelling constructs provided by the OntoWeaver site ontologies. In particular, we focus on the specification of complex navigation structures, user interfaces, and presentation instructions. In chapter 7, we will come back to this example and describe how to construct this web portal using OntoWeaver and its tool suite.

5.2 The site view ontology

The site view ontology abstracts navigational structures and user interfaces of data-intensive web sites. Figure 5.3 shows an overview of the site view ontology. It models a web site as a collection of logical resources. Logical resources are abstracted as compositions of resource components. Resource components are further described as compositions of a number of user interface elements, e.g., output elements, input elements, command elements, and sub resource components.

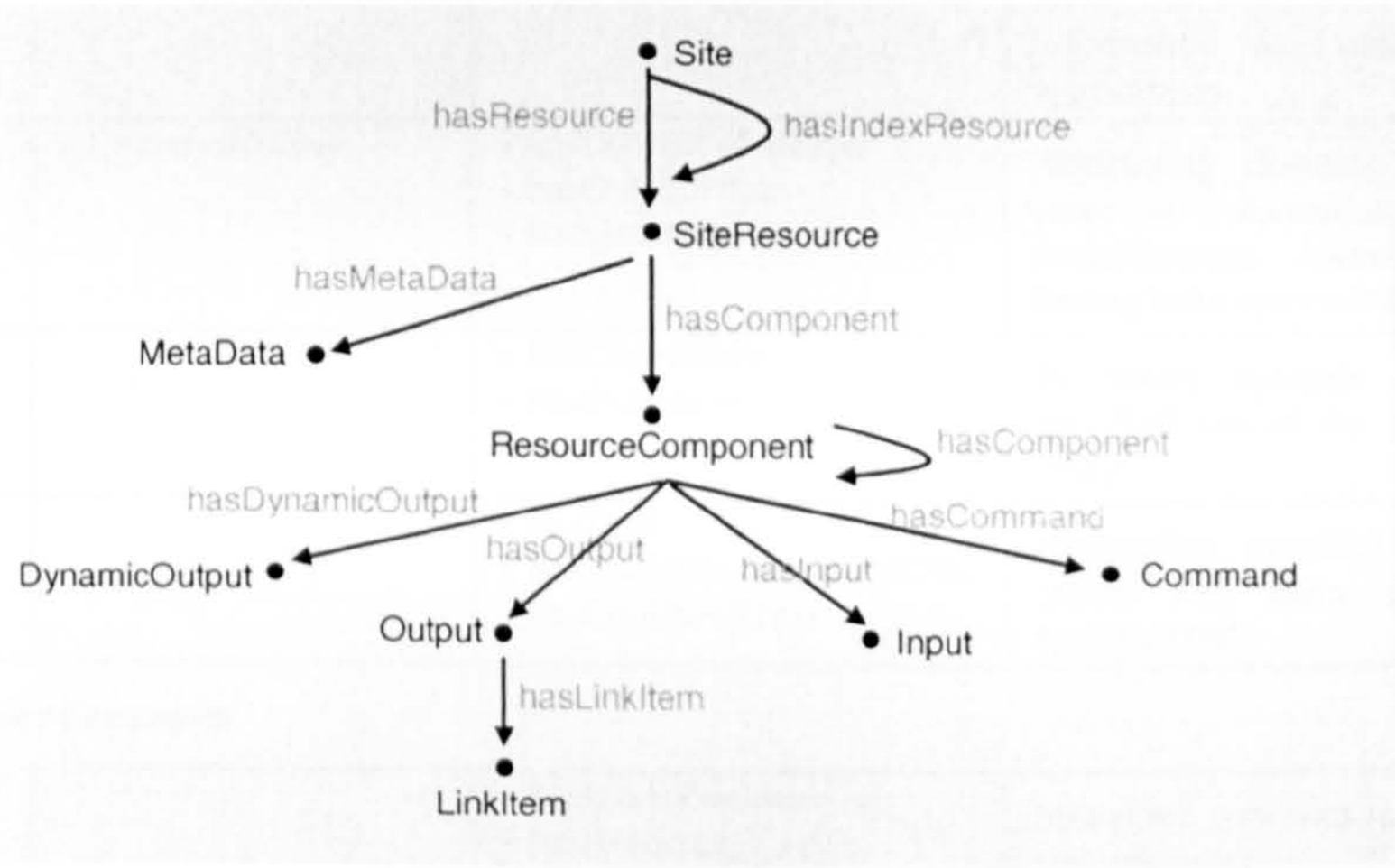


Figure 5.3 An overview of the site view ontology

Table 5.1 describes the major constructs of the site view ontology, which can be classified into three main categories. They are *navigational constructs*, *atomic user interface constructs*, and *composite user interface constructs*. The navigational constructs characterize complex navigation structures, including contextual links and non-contextual ones. The atomic user interface constructs describe those site view elements that can not be further decomposed. The composite user interface constructs express those site view elements that are typically composed by a number of sub elements. Apart from these three categories of constructs, there are also other constructs which support the specification of these main constructs, as shown in table 5.1.

Table 5.1 The major constructs of the site view ontology

Constructs	Sub-Constructs	Slots	Descriptions
Navigational Constructs			
LinkItem	• DynamicLinkItem • ContextualLinkItem	• hasAssociatedResourceURI	Modelling link relationships.
DynamicLinkItem		• hasClassEntity • hasSlotEntity	Modelling link items that come from the underlying knowledge bases.
ContextualLinkItem		• hasAssociatedResourceURI • hasInstanceConstraint	Describing link items that are contextual
Atomic User Interface Constructs			
Input		• hasClassEntity • hasSlotEntity	Expressing interface elements that allow end users entering meaningful information to interact with web applications.
Output	• DynamicOutput	• hasOutputValueType • hasOutputValue • hasLinkItem	Abstracting elements that present a piece of information, either being static/dynamic, text/image, plain or having links associated with it.
DynamicOutput		• hasClassEntity • hasSlotEntity	Presenting dynamic value for the specified slot of the associated class entity.
Command		• hasTask • hasAssociatedResourceURI • hasCommandText	Abstracting interface elements that enable end users to invoke the specified task.
Composite User Interface Constructs			
Site		• hasIndexResource • hasResource	Modelling web sites as a collection of web resources.
SiteResource		• hasComponent • hasMetaData	Modelling web resources.
ResourceComponent	• KAComponent • DataComponent • SearchComponent • InputComponent • OutputComponent	• hasComponent • hasOutput • hasInput • hasCommand	Modelling user interface elements that compose web pages.
KAComponent		• hasClassEntity • hasInputComponent • hasCommand	Modelling components that allow users to input facts to the underlying databases.
DataComponent		• hasClassEntity • hasInstanceConstraint • hasOutputComponent	Modelling components that present data coming from the underlying knowledge bases.
SearchComponent		• hasClassEntity • hasInputComponent • hasCommand	Describing components that allow users to make queries over the underlying knowledge bases.
OutputComponent		• hasOutput • hasDynamicOutput	Describing site view components that present dynamic domain data content retrieved from a specified slot of a specified domain class entity.
InputComponent		• hasOutput • hasInput	Describing site view components that gather input from end users for a particular slot of the specified class entity.
Others			

MetaData		• hasPageTitle • hasIntroduction • hasAssociatedClassEntity • hasAuthor • hasDescription • hasMetaString	Describing meta information for page nodes. The meta information can be used to describe rough information for pages and facilitate querying about web pages.
Task		• hasTaskDescription	Expressing tasks associated with site view components. OntoWeaver defines a number of task primitives which are associated with knowledge acquisition components and search components, including NEW-DATA-ENTRY, DATA-QUERYING and DATA-UPDATING.
InstanceConstraint		• hasConstrainedClassEntity • hasConstrainedSlotEntity • hasConstrainedRelation • hasConstrainedValue • hasLogicOperator	Formalizing constraints that can be placed upon instances
RelationOperator			Describing relation operators. OntoWeaver defines a number of operator primitives, including EQUAL, NOT-EQUAL, GREATER-THAN, NO-LESS-THAN, LESS-THAN, NO-GREATER-THAN, and PART-OF.
LogicOperator			Defining logic operators. Three primitives are defined, which supports the specification of logic operators. They include NOT, AND, and OR.

In addition to the support for the specification of navigation structures and the composition of user interfaces, these site view constructs also support the specification of i) static site view elements whose content is pre-defined at design time, ii) dynamic site view elements whose content is retrieved from the underlying domain ontology, and iii) interactive site view elements, which allow interactions of end users with web sites, such as supplying information or invoking specific services. In the following sub sections, we will illustrate these constructs.

5.2.1 Navigational constructs

Links play a crucial role in web sites as they are the major components supporting navigation. To specify links, URLs of the associated linked web pages are required, which can be pre-defined (i.e. *static* links) or retrieved from the underlying domain data layer (i.e. *dynamic* links). In the case of contextual links, contextual information needs to be specified to ensure correct information flow from the source pages to the linked web pages. The site view ontology distinguishes these three types of links and provides the following constructs to describe them:

- The construct *LinkItem* which abstracts static links in terms of *hasAssociatedResourceURI* specifying the URL of the linked web resource.
- The construct *DynamicLinkItem* which relies on slots *hasClassEntity* and *hasSlotEntity* to specify the source of the URL of the linked web resource. The following RDF code[1] defines a dynamic link in which the URL of the linked web resources is retrieved from the slot *web_address* of the class *Project*. The site view ontology references entities defined in the domain ontology by means of their Uniform Resource Identifiers (URI) [Berners-Lee et al., 1998]. In this example, the string "&do; Project" specifies that the associated class is the class *Project* defined in the specified domain ontology. The string "&do; web_address" indicates that the associated resource URI of the link in question is retrieved from the slot *web_address* of the specified class (Please note that to enable readability the URIs shown in this chapter are simplified).

```
<rdf:Description rdf:about="…/project-url-link" >
  <rdf:type rdf:resource="&svo;DynamicLinkItem"/>
  <svo:hasClassEntity rdf:resource="&do;Project" />
  <svo:hasSlotEntity rdf:resource="&do;web_address" />
</rdf:Description>
```

- The construct *ContextualLinkItem* which relies on a slot called *hasInstanceConstraint* to describe the associated contextual information constraining the data content presented in the linked web page in terms of the class *InstanceConstraint*. A contextual link example in the KMi web portal is the link in the project web page, which allows navigation to the web page to present the detailed information about the specified person. This link can be associated with different user interface elements which present the names of people relevant to each project instance, e.g., leaders or members. The following code illustrates the specification of this contextual link. The contextual information constrains the instances of the class *Person*, using the person name that an end user clicks on as the filter of the slot *person_name*. The filtering value is specified as "parent.value",

---

[1] The prefix 'svo' refers to the namespace of the site view ontology: xmlns:svo=http://kmi.open.ac.uk/people/yuangui/siteviewontology#. The prefix 'do' refers to the namespace of the underlying domain ontology of the target web site.

which means the value of the output element which visualizes the contextual link. As will be described in the following section, links rely on output elements to realize their visualization.

```
<rdf:Description rdf:about="…/project-member-contextual link" >
  <rdf:type rdf:resource="&svo;ContextualLinkItem" />
  <svo:hasAssociatedResourceURI>…/person_page.jsp</svo:hasAssociatedResourceURI>
  <svo:hasInstanceConstraint rdf:resource="#person-name constraint" />
</rdf:Description>
<rdf:Description rdf:about="…/person name-constraint" >
  <rdf:type rdf:resource="&svo;InstanceConstraint" />
  <svo:hasConstrainedClassEntity rdf:resource="&do;Person" />
  <svo:hasConstrainedSlotEntity rdf:resource="&do;person_name" />
  <svo:hasConstrainedRelation  rdf:resource="&svo;EQUAL"/>
  <svo:hasConstrainedValue>parent.value</svo:hasConstrainedValue>
</rdf:Description>
```

5.2.2 The atomic user interface constructs

OntoWeaver distinguishes three types of atomic user interface elements. They are *output elements* which present static or dynamic information pieces, *input elements* which allow end users to input information, and *command elements* which allow end users to invoke the associated services. Input elements and command elements are typically used in knowledge acquisition and data querying forms. OntoWeaver provides the following constructs to address these basic elements:

- The construct *Output* which models the output elements that present *static* information. It relies on slots *hasOutputValueType* and *hasOutputValue* to specify the presented information which can be a piece of text, an image, a video clip, or an audio clip. The presented information can be associated with links. Such association is described by means of the slot *hasLinkItem*.
- The construct *DynamicOutput* which expresses those output elements that present values of the specified slot of the associated class entity. It is a sub-class of the construct *Output*. It employs slots *hasClassEntity* and *hasSlotEntity* to indicate the source of the dynamic data that will be presented.
- The construct *Input* which describes input fields. Like the construct *DynamicOutput*, this construct employs slots *hasClassEntity* and *hasSlotEntity* to specify the concept that the information gathered from the input element corresponds to.

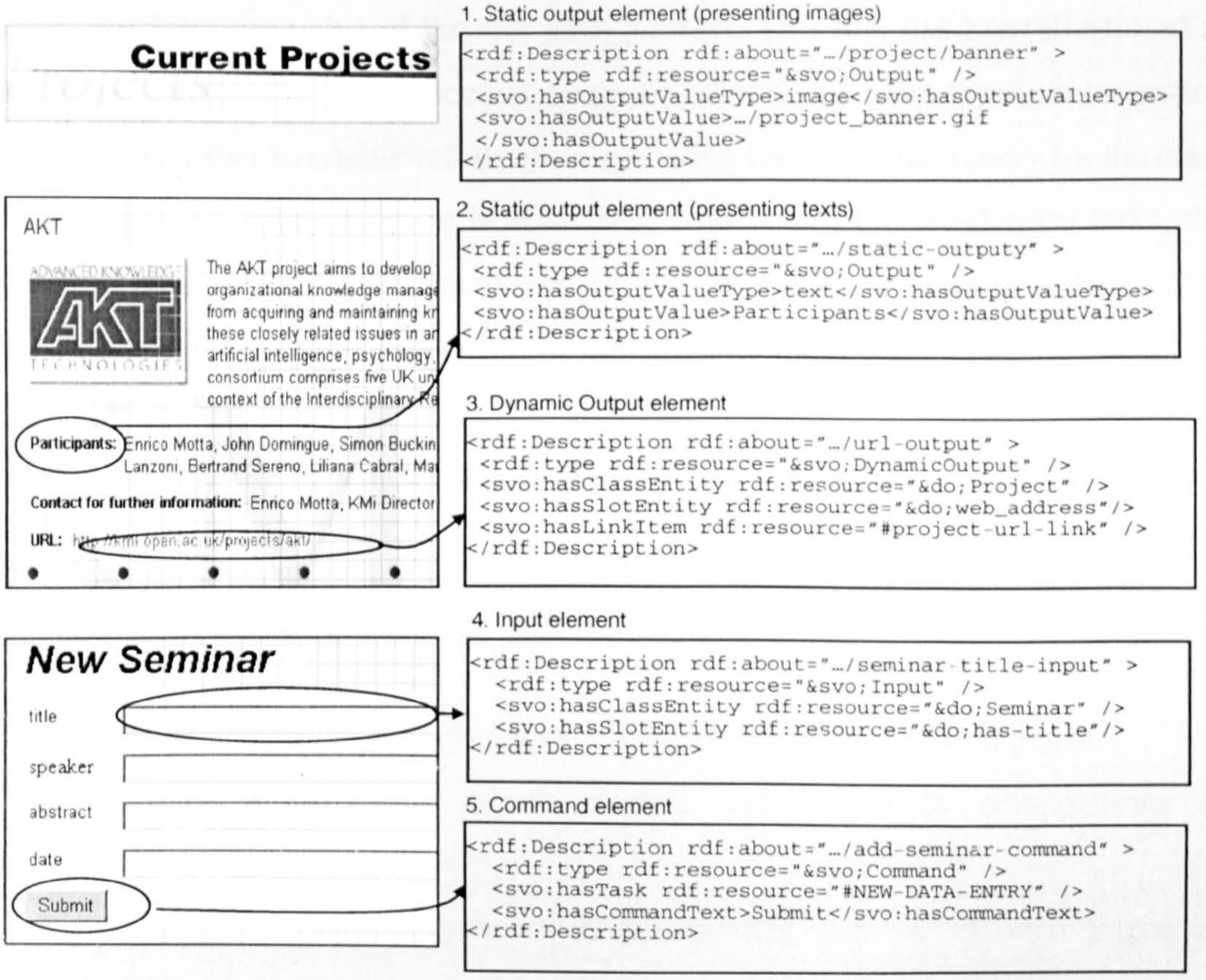


Figure 5.4 Examples of atomic user interface elements and their specifications

- The construct *Command* which abstracts command elements by means of slots *hasTask*, *hasAssociatedResourceURI*, and *hasCommandText*. The slot *hasTask* defines the associated task. OntoWeaver provides a set of built-in services for target web sites, such as data retrieving, data querying, and data acquisition to allow the access of the data layer. The slot *hasAssociatedResourceURI* specifies the web resource which will be presented after the invocation of the associated task. This associated web resource typically presents the results of the associated task. The slot *hasCommandText* indicates the text that will appear in the command element.

Figure 5.4 illustrates some examples of atomic user interface elements. Specifically, the first two examples illustrate how to specify elements for presenting static information (e.g. texts and images). The third example describes an element which presents values of the slot *web_address* for instances of the class *Project*. This

element is associated with a dynamic link in which the URL of the linked web page is retrieved from the value of the slot *web_address*. This link has been illustrated in the section above. The input element example allows end users typing information for the slot *has-title* to enable information gathering for a new data entry for the class *Seminar*. The command element is associated with one of the pre-defined tasks, the task *NEW-DATA-ENTRY*. As mentioned earlier, the site view ontology relies on the use of URI to reference entities defined in the domain ontology.

5.2.3 The composite user interface constructs

The composite user interface constructs model those site view elements which are composed of a number of sub elements. They include:

- The construct *Site* which models a web site as a composition of web pages.
- The construct *SiteResource* which models web pages as compositions of components.
- The construct *ResourceComponent* which models content of web pages as compositions of atomic user interface elements and sub components.
- A set of *component primitives* which model typical dynamic user interfaces of web pages. OntoWeaver distinguishes three kinds of typical user interfaces in data-intensive web sites which include user interfaces for data presentation, data acquisition, and data querying. OntoWeaver provides constructs *DataComponent*, *KAComponent*, and *SearchComponent* to address them accordingly. Specifically, the construct *DataComponent* relies on the slot *hasClassEntity* to specify the domain class and the slot *hasInstanceConstraint* to specify constraints to filter instances of the associated class. The construct *KAComponent* models user interface elements which allow the acquisition of data facts from end users for the specified domain class. The construct *SearchComponent* abstracts user interface elements which allow the querying of the domain data. Both of these two latter constructs rely on the slot *hasClassEntity* to indicate the associated domain class. As shown in table 5.1, they both contain a set of input components which allow the typing of information from end users and a command element which supports the gathering of information and the invocation of the associated services. The major

difference between them is that they are associated with different tasks. The knowledge acquisition components are associated with the knowledge acquisition task *NEW-DATA-ENTRY*, while the search components are associated with the data querying task *DATA-QUERYING*.

DataComponent

AKT

The AKT project aims to dev
organizational knowledge ma
from acquiring and maintainir
these closely related issues
artificial intelligence, psychol
consortium comprises five UI
context of the Interdisciplinar

Participants: Enrico Motta, John Domingue, Simon Bu
Lanzoni, Bertrand Sereno, Liliana Cabral

Contact for further information: Enrico Motta, KMi Dire

URL: http://kmi.open.ac.uk/projects/akt/

```
<rdf:Description rdf:about="…/project/datacomponent" >
 <rdf:type rdf:resource="&svo;DataComponent" />
 <svo:hasClassEntity rdf:resource="&do;Project" />
 <svo:hasOutputComponent
      rdf:resource="…/outputcomponent/project_name"/>
 …
</rdf:Description>

<rdf:Description rdf:about="…/outputcomponent/project_name" >
 <rdf:type rdf:resource="&svo;OutputComponent" />
 <svo:hasDynamicOutput rdf:resource="…/dynamic/project_name"/>
</rdf:Description>

<rdf:Description rdf:about="…/dynamic/project_name" >
  …
</rdf:Description>
…
```

KAComponent

New Seminar

title

speaker

abstract

date

Submit

```
<rdf:Description rdf:about="…/seminar/ka" >
 <rdf:type rdf:resource="&svo;KAComponent" />
 <svo:hasClassEntity rdf:resource="&do;Seminar" />
 <svo:hasInputComponent
      rdf:resource="…/inputcomponent/title"/>
 <svo:hasCommand rdf:resource="…/add-seminar-command"/>
</rdf:Description>

<rdf:Description rdf:about="…/inputcomponent/title" >
 <rdf:type rdf:resource="&svo;InputComponent" />
 <svo:hasInput rdf:resource="…/similar-title-input" />
 <svo:hasOutput rdf:resource="…/similar-title-output" />
</rdf:Description>
…
```

Figure 5.5 Examples of composite user interface elements and their specifications

Figure 5.5 shows samples of user interfaces for data presentation and acquisition and their specifications in terms of the constructs provided in the site view ontology. The data component presents instances of the class *Project*. It relies on the constructs *DataComponent*, *OutputComponent*, *DynamicOutput*, and *Output* to specify its compositional structure of the content. The knowledge acquisition component allows the acquisition of new facts of the class *Seminar* from end users. As shown in the figure, constructs *KAComponent*, *InputComponent*, *Input*, and *Command* are used to specify this user interface.

5.2.4 Other constructs

To support the specification of the main constructs described above, OntoWeaver provides a number of support constructs. Examples include the construct *MetaData*

which abstracts meta-data for web resources, the construct *Task* which allows the specification of associated tasks for site view elements, the construct *InstanceConstraint* which supports the specification of instance constraints for classes, the construct *RelationOperator* which supports the specification of relation operators, and the construct *LogicOperator* which allows the specification of logic operators. Furthermore, as shown in table 5.1, a number of primitives are defined for the constructs *Task* (describing built-in tasks), *RelationOperator* (expressing relation operators), and *LogicOperator* (describing logic operators).

5.2.5 User interface composition

The site view ontology relies on the composite constructs (e.g., *SiteResource* and *ResourceComponent*) and *the atomic user interface constructs* to realize a mechanism which allows the composition of complex user interfaces. Specifically, the user interfaces of web pages are composed of a number of resource components. Each resource component further contains atomic user interface elements and sub resource components. Thus, a complex user interface can be composed. In the following, we will use a number of examples to illustrate the user interface composition mechanism.

Example 1: Specifying site structures

A site structure in OntoWeaver comprises an index page node which defines an entry point for the web site and a number of page nodes which form the navigation space. OntoWeaver relies on the construct *Site* to describe the composition of web sites, the construct *SiteResource* to specify each page node, and the construct *LinkItem* to express link relations between page nodes. The following RDF code fragment illustrates the composition of the site structure of the KMi web portal.

```
<rdf:Description rdf:about="…/kmi_portal">
    <rdf:type rdf:resource="&svo;Site" />
    <svo:indexResource rdf:resource="…/homepage"/>
    <svo:siteResource>
       <rdf:Bag>
         <rdf:li rdf:resource="…/project-page "/>
         <rdf:li rdf:resource="…/kmi_member-page "/>
         <rdf:li rdf:resource="…/grants "/>
         …
       </rdf:Bag>
```

```
    </svo:siteResource>
</rdf:Description>
```

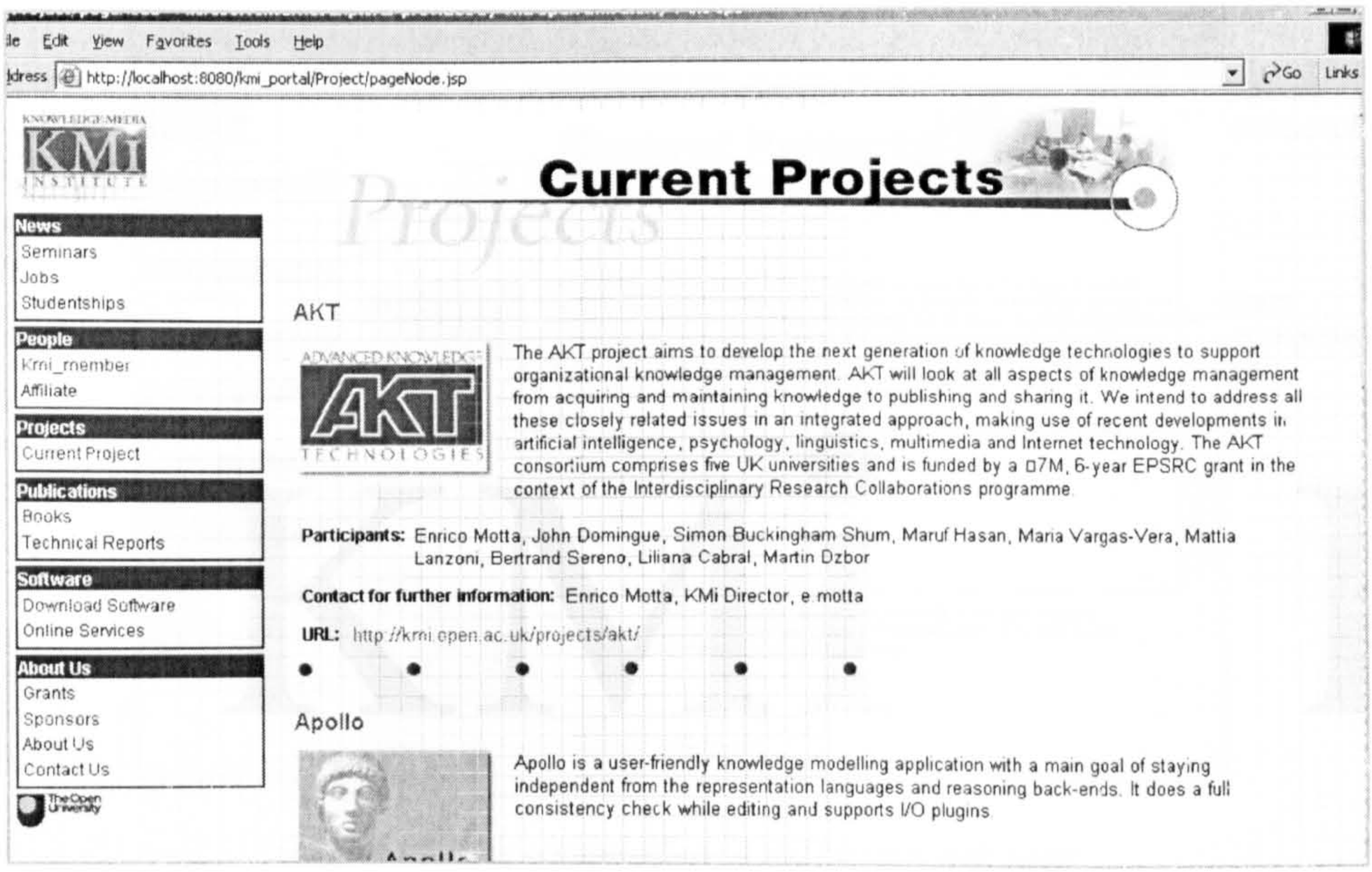


Figure 5.6 A screenshot of the project web page which presents instances of the class *Project*

Each page node in the site structure is defined as an instance of the construct *SiteResource*. At the coarse-grained level, the purpose of each page node is captured by means of the construct *MetaData*. The link relations between page nodes are defined within the specification of page nodes in terms of the navigational constructs. In addition to meta-data and navigational links (which can be placed in one component or scattered in a number of components), page nodes also comprise components which form other contents apart from navigation paths. Figure 5.6 shows a page node example which presents instances of the class *Project*. The following RDF code fragment illustrates the specification of this page node. The detailed composition of user interfaces will be illustrated in the next example.

```
<rdf:Description rdf:about="…/project-page" >
    <rdf:type rdf:resource="&svo;SiteResource"/>
    <svo:hasMetadata rdf:resource=="…/project-page/metadata" />
    <svo:hasComponent>
      <rdf:Bag>
        <rdf:li rdf:resource="…/navigationcomponent"/>
        …
      </rdf:Bag>
  </svo:hasComponent>
</rdf:Description>

<rdf:Description about="…/project-page/metadata" >
  <rdf:type rdf:resource="&svo;MetaData"/>
  <svo:hasPageTitle>Projects in the Knowledge Media Institute (KMi)</svo:pageTitle>
  <svo:hasAssociatedClassEntity rdf:resource="&do;Project"/>
```

```
    ...
</rdf:Descripltion>
```

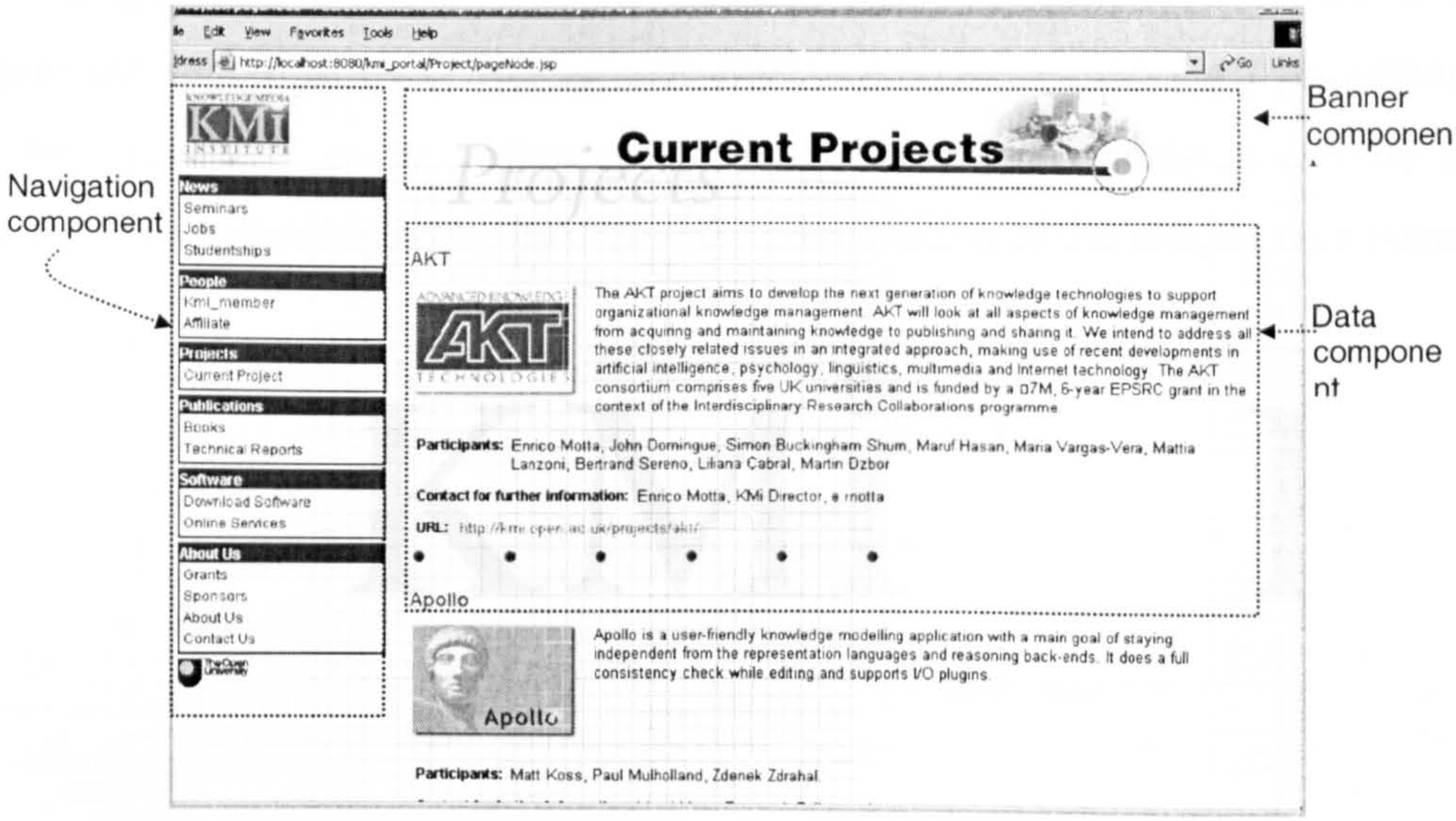


Figure 5.7 The composition of the project web page

Example 2: composing user interfaces for web pages

As illustrated in figure 5.7, the user interface of the project web page is composed of three components: *a navigation component* which presents hyperlinks, *a banner component* which displays a banner for the web page, and *a data component* which presents instances of the class *Project*. Each component is further made up of a number of sub-elements. Specifically, the navigation component is composed of a number of sub components. Each sub component further contains a number of output elements which are associated with links to other web pages. The banner component comprises one output element which presents a banner picture for the web page. The data component is composed of a number of sub elements as illustrated in figure 5.5.

As studied in chapter 2, current approaches are only able to express a fixed number of typical user interfaces in terms of the provided primitives. Composing site views is not supported. Hence, they can not provide appropriate support for the specification of the example user interface described above at the conceptual level. OntoWeaver on the other hand provides comprehensive support. Firstly, OntoWeaver provides atomic user interface constructs to capture those elements which can not be decomposed, such as output elements, input elements, and

command elements. Secondly, it provides composite constructs which allow the assembling of user interface elements. Thirdly, it provides primitive constructs which support the specification of typical user interface elements and most importantly as will be illustrated in the following example such user interfaces can be easily adjusted. The following code illustrates the composition of the sample user interface.

```
  <!-- the user interface of the web page is composed of a set of components -->
<rdf:Description rdf:about="…/project-page" >
  <rdf:type rdf:resource="&svo;SiteResource" />
  <svo:hasMetadata rdf:resource="…/project-page/metadata" />
  <svo:hasComponent>
    <rdf:Bag>
       <rdf:li rdf:resource="…/navigationcomponent"/>
       <rdf:li rdf:resource="…/project/bannercomponent"/>
       <rdf:li rdf:resource="…/project/datacomponent"/>
    </rdf:Bag>
  </svo:hasComponent>
</rdf:Description>
  <!-- the navigation component comprises a number of sub components -->
<rdf:Description rdf:about="…/navigationcomponent" >
  <rdf:type rdf:resource="&svo;ResourceComponent" />
  <svo:hasComponent>
     <rdf:Bag>
       <rdf:li rdf:resource="…/Newscomponent"/>
       …
       <rdf:li rdf:resource="…/Aboutuscomponent"/>
    </rdf:Bag>
  </svo:hasComponent>
</rdf:Description>
…
  <!-- the news component is composed of a number of output elements -->
<rdf:Description rdf:about="…/Newscomponent" >
  <rdf:type rdf:resource="&svo;ResourceComponent" />
  <svo:hasOutput>
       …
  </svo:hasOutput>
</rdf:Description>
…
<rdf:Description rdf:about="…/project/bannercomponent" >
  <rdf:type rdf:resource="&svo;ResourceComponent" />
  <svo:hasOutput rdf:resource="="…/project/banner" />
</rdf:Description>
…
```

Example 3: adapting typical user interfaces of data-intensive web sites

As studied earlier in chapter 2, current approaches do not support the adaptation of typical user interfaces, due to the lack of expressive user interface models. Now we investigate how OntoWeaver addresses this problem. We use the user interface of data components as an example. As shown in part (a) of figure 5.8, the default user interface of a data component is composed of a number of dynamic output elements presenting values of slots for instances of the specified class and a number of static output elements presenting explanations about the dynamic values. Each sub element is specified declaratively and available for modification. Furthermore, new elements can be easily added into the user interface, as the construct *DataComponent* supports the flexible assembling of user interface elements. Part (b) of figure 5.8 shows an adapted user interface example: the static output elements for presenting

explanations about the values of slots *project_name*, *picture*, and *description* have been removed, as their explanations can be indicated by their content or presentation styles. The output type of the dynamic output *picture* has been changed from text to image. Apart from the content, the presentation styles and layouts have also been adapted. This will be explained in the following section.

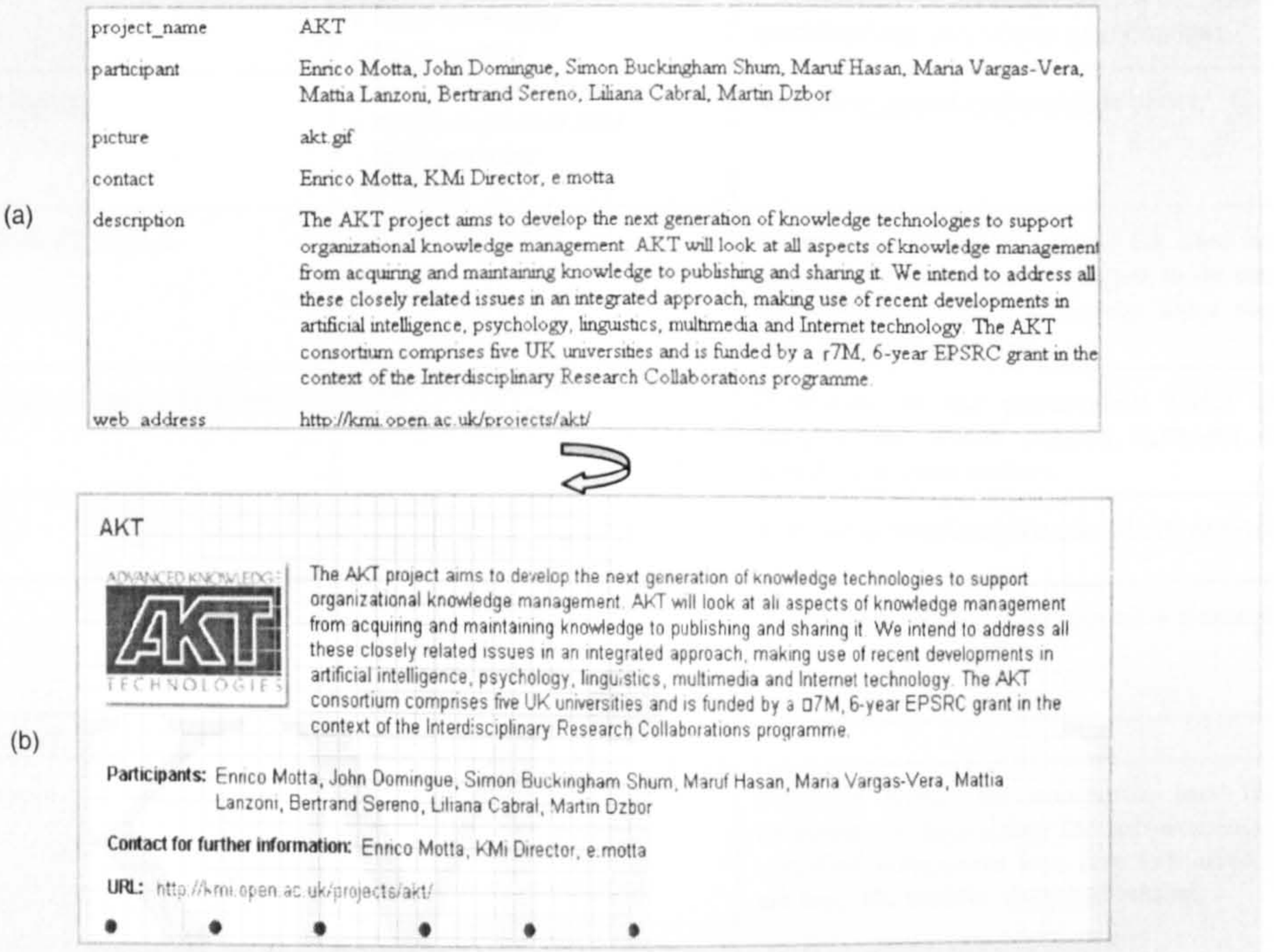


Figure 5.8 An example of adapting typical user interfaces. Part (a) shows the default user interface of the project data component. Part (b) shows the adapted user interface.

5.3 The presentation ontology

The presentation ontology provides explicit vocabularies to allow the specification of presentation instructions for the target web site. As shown in figure 5.9, the presentation ontology describes a presentation instruction of the target web site as a collection of *templates*, *presentation objects*, and *layout objects*. Templates describe presentation styles e.g. backgrounds, colours, and fonts. Presentation objects specify templates for user interface elements. Layout objects express organization instructions. The reference of user interface elements in the presentation model is

realized by means of their Uniform Resource Identifiers (URIs). Table 5.2 describes the major constructs of the presentation ontology.

Table 5.2 The major constructs of the presentation ontology

Construct	**Slots**	**Description**
Presentation Constructs		
SitePresentation	• hasSiteURI • hasLayout • hasPresentation • hasTemplate	Describing a web site presentation model as a collection of template specifications, presentation specifications, and layout specifications.
PresentationTemplate	• hasBackgroundImage • hasBackgroundColor • hasForeColor • hasFont	Modelling generic presentation styles.
WidgetPresentationTemplate	• hasWidgetType	Modelling presentation styles for user interface elements, which require widgets to be rendered, e.g. dynamic output elements, input elements, and command elements.
DataComponentPresentationTemplate	• hasColumnCount • hasRowsPerPage	Dedicating to the presentation styles of data components, which publish instances of the specified domain entities.
Presentation	• hasTemplateURI • hasSiteEntityURI	Specifying templates for user interface elements.
Font	• hasFontColor • hasFontFamily • hasFontSize • hasFontStyle	Describing font styles for site view elements
Layout Constructs		
ComponentLayout	• hasSiteEntityURI • hasTopAreaLayout • hasLeftAreaLayout • hasMiddleAreaLayout • hasRightAreaLayout • hasBottomAreaLayout	Defining layout for composite user interface elements, by organizing the sub-elements of the specified component into five sub areas, which are top, left, middle, right and bottom.
TextLayout	• hasAlignment	Specifying alignment for atomic user interface element.
ComponentAreaLayout	• hasSiteEntityURI • hasLayoutDirection • hasAreaSize	Modelling layout for the specified area in the composite component.
AreaSize	• hasSizeType • hasWidth • hasHeight	Defining size for a particular area which is typically contained in composite user interface elements.
Others		
FontStyle		Expressing styles of fonts, e.g. bold, italic, etc.
WidgetType		Specifying types of widgets for rendering site view elements. A number of primitives are defined, including NONE-WIDGET, TEXT-FIELD, TEXT-AREA, BUTTON, and IMAGE.
TextAlignment		Describing the alignment of text, e.g., left, center, right, etc.
AreaLayoutDirection		Expressing the direction (e.g., horizon or vertical) of placing site view elements in a sub-area of composite site view elements. Two primitives are defined for this construct. They are HORIZONTAL-DIRECTION, and VERTICAL-DIRECTION.

AreaSizeType		Defining the type of the specified size. Two primitives are defined, including RELATIVE and ABSOLUTE.

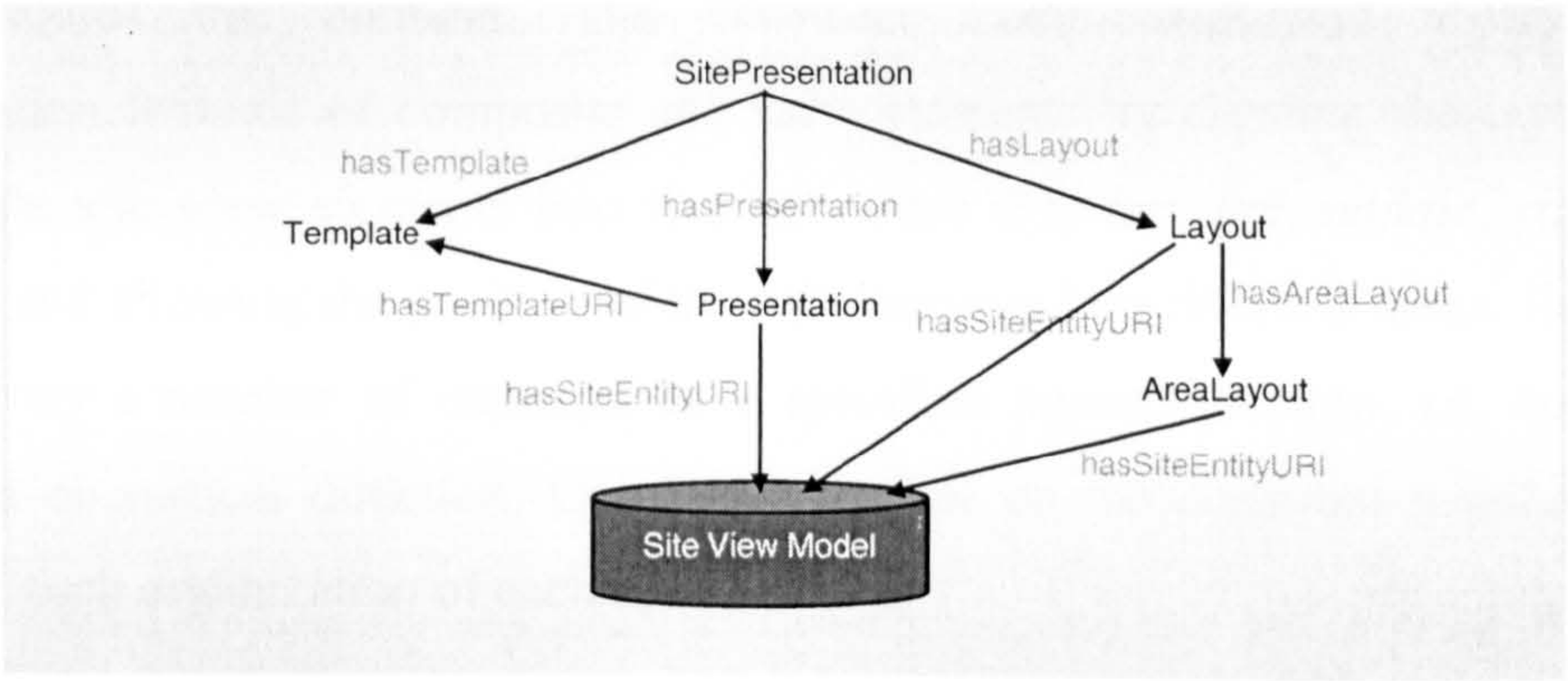


Figure 5.9 An overview of the presentation ontology

By using templates, presentation styles can be specified once and re-used as many times as possible. Furthermore, it allows consistent presentation instructions to be specified for specific site view elements. As will be described later in section 5.5.3, the cascading stylesheets (CSS) approach and the extensible style sheet language (XSL) have their own specific targets. They can not be re-used in OntoWeaver.

OntoWeaver distinguishes three types of presentation styles and provides corresponding template constructs to describe them. They include the construct *GenericPresentationTemplate* which describes generic presentation styles shared in most user interface elements (e.g., background, colours, and fonts), the construct *WidgetPresentationTemplate* which describes presentation styles for user interface elements where widgets are involved (e.g., dynamic output elements, input elements, and command elements), and the construct *DataComponentPresentationTemplate* which specifies presentation styles for data components presenting dynamic data content. In order to allow the attachment of templates to site view elements, OntoWeaver provides a construct called *Presentation* which relies on the slot *hasSiteEntityURI* to reference the working site view element and the slot *hasTemplateURI* to specify a template object.

Two layout constructs are provided to model organizations of site view elements. The construct *TextLayout* models the layout of atomic site view elements in terms of *alignment*, specifying the alignment of a user interface element within the component that contains this element. The construct *ComponentLayout* abstracts the organization features of composite site view elements by dividing the layout of a composite site view elements into five sub areas (i.e. *top*, *left*, *middle*, *right*, and *bottom*) and allowing the position of the sub-elements into different areas. Each area can display a number of elements in a specified layout direction, i.e. horizontal direction or vertical direction. OntoWeaver relies on the construct *AreaLayout* to describe such organization of each area.

Now we investigate how the presentation ontology facilitates the specification of presentation instructions for the target web site. We use the sample user interfaces shown in figure 5.6 as a study case. A number of templates have been defined for rendering the user interface elements. For example, a template has been defined for the dynamic output element that presents values of the slot *project_name*, which renders the values of project titles in a bold font with a slightly large size with no widget involved. The following RDF code[2] illustrates how to specify such a template and to attach it with the user interface element.

```
<rdf:Description rdf:about="…/template1" >
  <rdf:type rdf:resource="&spo;WidgetPresentationTemplate" />
  <spo:hasWidgetType rdf:resource="&sco;NONE-WIDGET" />
  <spo:hasFont rdf:resource="…/template1/font" />
  …
</rdf:Description>

<rdf:Description rdf:about="…/template1/font" >
  <rdf:type rdf:resource="&spo;Font" />
  <spo:hasFontStyle rdf:resource="&spo;BOLD" />
  …
</rdf:Description>

<rdf:Description rdf:about="…/presentation1" >
  <rdf:type rdf:resource="&spo;Presentation" />
  <spo:hasTemplateURI> …/template1 </spo:hasTemplateURI>
  <spo:hasSiteEntityURI>…dynamic/project_name</spo:hasSiteEntityURI>
</rdf:Description>
```

Regarding the layout of the sample user interface, the sub component *project_name* is placed at the top; the sub component *picture* is arranged at the left, the sub

---

[2] The prefix 'spo' refers to the namespace of the site presentation ontology: xmlns:spo=http://kmi.open.ac.uk/people/yuangui/sitepresentationontology#

component *description* is put at the middle; and other sub-components are organized at the bottom vertically. This layout can be easily specified by means of the OntoWeaver layout constructs. The following fragment of RDF code illustrates the layout specification of the data component which only arranges the top-level sub-components. Each sub component can have its own layout design. Thus, a complex layout can be specified for a user interface element.

```
<rdf:Description rdf:about="…/componentlayout1" >
  <rdf:type rdf:resource="&spo;ComponentLayout" />
  <spo:hasSiteEntityURI>…/project/datacomponent</spo:hasSiteEntityURI>
  <spo:hasTopAreaLayout rdf:resource="#componentlayout1_toparea" />
  <spo:hasLeftAreaLayout rdf:resource="#componentlayout1_leftarea" />
  <spo:hasMiddleAreaLayout rdf:resource="#componentlayout1_middlearea" />
  <spo:hasBottomAreaLayout rdf:resource="#componentlayout1_bottomarea" />
</rdf:Description>

  <!-- The component project_name is placed in the top area. -->
<rdf:Description rdf:about="…/componentlayout1_titlearea" >
  <rdf:type rdf:resource="&spo;AreaLayout"/>
  <spo:hasSiteEntityURI>
    <rdf:Bag>
       <rdf:li>…/project/datacomponent/project_name</rdf:li>
    </rdf:Bag>
  </spo:hasSiteEntityURI>
</rdf:Description>

    <!-- The component picture is placed in the left area. -->
<rdf:Description rdf:about="…/componentlayout1_leftarea" >
  <rdf:type rdf:resource="&spo;AreaLayout"/>
  <spo:hasSiteEntityURI>
    <rdf:Bag>
       <rdf:li>…/project/datacomponent/picture</rdf:li>
    </rdf:Bag>
  </spo:hasSiteEntityURI>
</rdf:Description>

    <!-- The component description is placed in the middle area. -->
<rdf:Description rdf:about="…/componentlayout1_middlearea" >
  <rdf:type rdf:resource="&spo;AreaLayout"/>
  <spo:hasSiteEntityURI>
    <rdf:Bag>
       <rdf:li>…/project/datacomponent/description</rdf:li>
    </rdf:Bag>
  </spo:hasSiteEntityURI>
</rdf:Description>

   <!-- The components project_member and contact are placed in the bottom area vertically.-->
<rdf:Description rdf:about="…/componentlayout1_bottomarea" >
  <rdf:type rdf:resource="&spo;AreaLayout"/>
  <spo:hasSiteEntityURI>
    <rdf:Bag>
       <rdf:li>…/project/datacomponent/project_member</rdf:li>
       <rdf:li>…/project/datacomponent/contact</rdf:li>
       …
    </rdf:Bag>
  </spo:hasSiteEntityURI>
  <spo:hasLayoutDirection rdf:resource="&spo;VERTICAL-DIRECTION" />
</rdf:Description>
```

5.4 Generating web site implementations from site ontologies

The OntoWeaver approach relies on the site ontologies to allow the declarative representation of data-intensive web sites. It makes possible for the design and development to be carried out at the conceptual level, in particular without having to rely on ad hoc approaches in the creation of complex user interfaces and visual appearances. In this section, we demonstrate that conceptual specifications of web

sites in terms of the OntoWeaver site ontologies can be compiled into implementations without the need of human intervention.

As described earlier in chapter 4, the OntoWeaver tool suite provides a tool called *Site Builder* to support this task. Figure 5.10 shows the process of the site generation. The Site Builder comprises an implementation code library which contains a set of implementation code templates for each construct of the site ontologies. It generates web site implementations by using the specifications of site views and presentations to instantiate the implementation code templates. The compiling process starts with scanning the site structure specified in terms of the construct *Site* and finding out all the page nodes contained in the target web site. It then works on compiling each page node into web page implementations. During the compiling process, the presentation instruction specification is also investigated if there is one specified. Otherwise, a default presentation instruction will be applied.

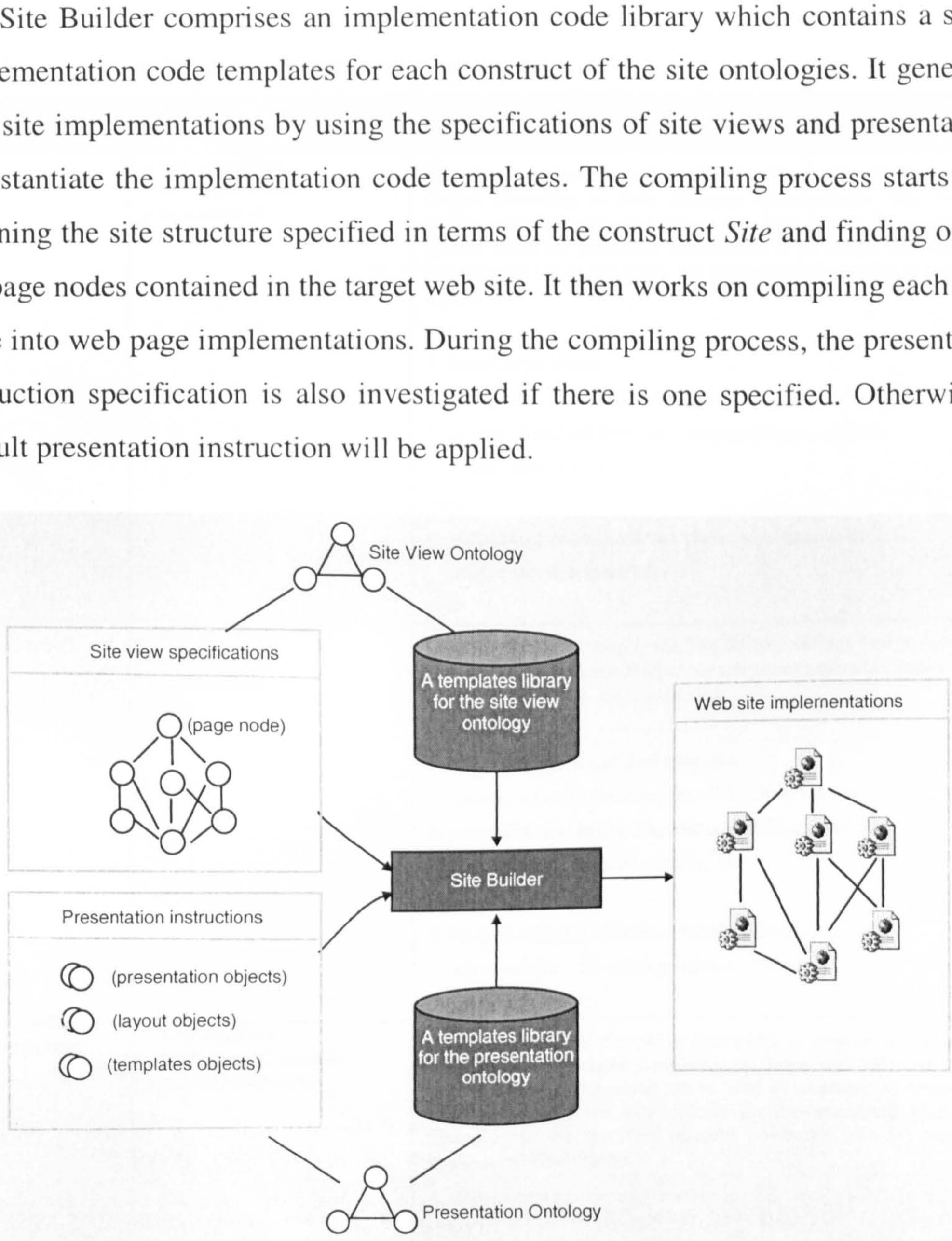


Figure 5.10 The process of generating web site implementations from ontologies

In the tool *Site Builder*, we use the Java Server Page (JSP) technology as the web site implementation technology. Thus, a set of JSP code templates have been defined.

Table 5.3 illustrates some examples of these code templates. The identifiers which begin with '&' are variables. Their values come from the attributes of the user interface elements or from the computation result. The sign of "<%" and "%>" indicates that the wrapped code is server side code, which is executed by web servers at run time.

Table 5.3 Examples of JSP code templates for site view constructs

Construct	**Slots**	**JSP code templates**
Output	• hasOutputType • hasOutputValue • hasLinkItem	Four templates have been defined in the Site Builder for the construct Output according to their different specifications. The first one applies when the presented information type is text. The second type applies when the presented information is an image. The third and fourth types are in use when the presented information is associated with links. ---------- 1. &hasOutputValue 2. <image src=&hasOutputValue> 3.<a:href=&hasLinkItem.hasAssociatedResourceURI> &hasOutputValue 4. <a:href=&hasLinkItem.hasAssociatedResourceURI> <image src=&outputURL>
Dynamicoutput	• hasClassEntity • hasSlotEntity • hasLinkItem	Like the construct Output, the Site Builder defines four templates for the constructs DynamicOutput, which present dynamic values for the specified slot of the associated class. ---------- 1. <% =bean.get(&hasSlotEntity) %> 2. <image src=<%=bean.get(&hasSlotEntity) %> > 3. <a:href=&hasLinkItem.hasAssociatedResourceURI> <%=bean.get(&hasSlotEntity) %> 4. <a:href=&hasLinkItem.hasAssociatedResourceURI> <image src= <%=bean.get(&hasSlotEntity) %>>

DataComponent	• hasClassEntity
	• hasInstanceConstraint
	• output components
		This template code comprises four parts: i) gets an instance of the run-time tool Database Connector, ii) maps the specified instance constraint to a string which can be used as parameter to retrieve data objects, iii) retrieves data objects for the associated class entity according to the specified instance constraint, and iv) repeatedly presents the data objects.
		---------- 	
		<%
		//part I – getting an instance of the run-time tool
		//Database Connector
		DatabaseConnector bean=session.get("databasebean");
		…
		//Part II - mapping the specification of the instance constraint to a string stored in the variable parameterString.

		String parameterSting=
		&hasInstanceConstraint.hasConstrainedSlot+
		&hasInstanceConstraint.hasConstrainedOperator+
		&hasInstanceConstraint.hasConstrainedValue
		// part III- retrieving instances
		bean.filter(
		&hasClassEntity, parameterString);
		
		// part IV – presenting instances
		while (bean.hasnext())
		{
		//display data repeatedly
		%>
		...
		<%} %>

KAComponent	• hasClassEntity
	• hasInputComponent
	• hasCommad
		This template code contains two parts. The first part presents a form which allows the gathering of input data. The second part gathers input data from the specified input fields and inserts the newly created instance into the populated domain database.
		--- 	
		// Part I – an HTML form
		<form action="ka.jsp">
		<!-- input form -->
		</form>
		
		// Part II – code for gathering input data from the HTML
		// form and invoking the task NEW-DATA-ENTRY of the
		// run time tool Database Connector
		<%
		//collecting information from the input form
		//invoking the specified service
		%>

SearchComponent	• hasClassEntity
	• hasInputComponent
	• hasCommand
		Like the template code of the construct KAComponent, this template code comprises two parts. The major difference is that the second part gathers input from end users and formulates a querying string and queries the underlying domain databases.
		--- 	
		<form action=
		"searchresultpage.jsp">
		<!-- input form -->
		</form>
		
		// searchresultpage.jsp
		<%
		//collecting information from the input form
		//invoking the query service of the run-time tool
		%>

Input	• hasClassEntity • hasSlotEntity	The Site Builder defines two template codes for the construct Input, which present input widgets to end users according to the specification of the corresponding presentation styles. One widget template is for typing short text and the other one is for typing long text.

		---------- 1. <Input type=&hasInputType name=&hasInputURI> 2. <TextArea type=&hasInputType name=&hasInputURI> </TextArea>
Command	• hasAssociatedResourceURI • hasTask • hasCommandText	The construct Command is mapped to the action part of a form and a built-in service, such as creating new instances or data querying. ---------- <form action=&hasAssociatedResourceURI> <!-- input form --> <INPUT type="submit" name=&commandURI value=&hasCommandText > </form>

SiteResource	• hasMetaData
	 • hasComponent
		This template code assembles the codes of all the sub component of the web page together.
		 ---------- 	
		 <html>
		 <title>&hasMetaData.hasPageTitle</title>
		 <meta name="description"
		 content=&hasMetaData.hasDescription />
		 // codes generated from component 1
		 ...
		 //codes generated from component n
		 ...
		 </html>

5.4.1 Compiling site view specifications

As shown in the table, the site view elements rely on the run-time tool *Database Connector* to allow the access and manipulation of the underlying domain database. The composite site view constructs which further comprises a set of sub elements do not have strict templates. This is true even for the component primitives: *data components*, *knowledge acquisition components*, and *search components*. This is because of the flexibility of their content. Web site developers can specify and edit the content of these components according to their particular requirements. Nevertheless, dynamic page codes can be generated by means of assembling the instantiated template code of each sub user interface element together. The process is however complex as some dynamic aspects can only be determined at compile time. For example, a knowledge acquisition component can have a variable number of input fields, and this cannot be dealt with simply through a simple form with a fixed number of entries. Server-side code for gathering information from the input fields

correctly and meaningfully is required. The OntoWeaver Site Builder generates a unique piece of JSP code file for each knowledge acquisition component to solve this problem.

Figure 5.11 shows an example of the generation of implementation code from the specification of the project data component (which presents instances of the class *Project* and has been explained in earlier in section 5.3). To enable readability, the implementation code has been simplified. The generated code retrieves the instances of the specified class and iterates the values of the retrieved instances. Only the values of those slots which have been chosen to appear in the data component will be iterated and presented in the data component.

```
<%
DatabaseConnector bean=session.get("databasebean");
String classEntity="Project";

//there is no instance constraint defined.
String parameterString="";

 //retrieve the instances of the class Project
bean.filter(classEntity, parameterString);
  //repeatedly presenting the instances of the class Project
while (bean.hasNext())
{ %>
  <table>
    …
   <% =bean.get("title") %>
   <image src=<% =bean.get("picture")%> >
   …
   participants: <% =bean.get("project_name") %>
   …
</table>
```

Figure 5.11 A fragment of JSP code generated from the specification of the project data component

5.4.2 Compiling presentation specifications

Apart from compiling the specified site view model, the Site Builder also investigates the specified presentation instructions and augments them into web site implementations. Different presentation style constructs have different code templates. For example, the generic presentation styles are mapped to the object *Style* of HTML; the data component presentation styles specified by the construct *DataComponentPresentationTemplate* are realized through the OntoWeaver run-time tool *Database Connector*; and the component layout organization is implemented by

relying on the HTML *table* object to position components in different areas according to the definitions. The following HTML code shows the template of the construct *ComponentLayout*, which makes use of table objects to divide five areas to place sub components of the associated composite site view component. In each area, the code generated for the specified components is placed according to the specified layout direction.

```
<table>
  <tr>
    //top area
  </tr>
  <tr>
     <table>
          <tr>
              <td>
                 //left area
              </td>
              <td>
                 //middle area
              </td>
              <td>
                  //right area
              </td>
          </tr>
     </table>
  </tr>
  <tr>
     //bottom area
  </tr>
</table>
```

5.5 Related work

In this section, we first compare our work to the closely related web modelling approaches with respect to support for the creation of complex views for data-intensive web sites. We then relate it to existing approaches on user interface modelling, presentation modelling, using ontologies to drive web site generation, and reference models.

5.5.1 Related work on conceptual web modelling

As described in chapter 2, a number of web modelling approaches have been developed to offer high level support for the design and development of data-intensive web sites. Examples include RMM [Isakowitz et al., 1995], OOHDM [Schwabe & Rossi, 1998], ARANEUS [Atzeni et al., 1998], HDM-lite [Fraternali & Paolini, 1998], Strudel [Fernandez et al., 1998], WSDM [De Troyer & Leune, 1998],

UWE [Hennicker & Koch, 2000], OO-H [Gómez et al., 2000], WebML [Ceri et al., 2000], OntoWebber [Jin et al., 2001], and Hera [Frasincar et al., 2002]. Despite the fact that these approaches share a common goal of supporting the activities required for web applications design, each approach has its own special focus. For example, RMM focuses on the design of domain data models and the design of navigational structures; Strudel concentrates on representing the underlying data structures as data graphs and on specifying site structures by means of queries over the data graphs. In this section we briefly compare OntoWeaver to current web modelling approaches with respect to conceptual level support for the specification of data-intensive web sites. We use the architecture clarified in chapter 4 to carry out the comparison. The architecture describes data-intensive web sites as a composition of a data layer, a site view layer, a presentation layer, and a customization layer. For details about this architecture, please refer to chapter 4.

Support for the design of the data layer

Current approaches support the abstraction of domain data structures and the specification of user interfaces for accessing the underlying domain data. Most approaches employ the Entity-Relationship (E-R) model and its variants to describe the data layer of data-intensive web sites. Some approaches, e.g., Strudel and OntoWebber, provide facilities to support the presentation of the underlying heterogeneous data coming from diverse data sources. Our work on the other hand focuses on developing ontologies to model the other layers of data-intensive web sites. It does not provide facilities to support the integration of data sources. But this will be our work in the future.

Support for the design of the site view layer

As studied in chapter 2, earlier approaches like HDM, RMM, OOHDM, HDM-lite, WSDM, and UWE only provide methodological support for the specification, due to the lack of explicit formal meta-models. Although recent approaches like WebML, OntoWebber, and Hera have overcome this problem, only coarse-grained primitives are provided to model typical user interface elements of web pages, such as elements

for data presentation, data querying, and data acquisition. As a consequence, web developers are only able to express a fixed number of typical user interfaces in terms of the provided primitives. The creation of complex user interface is out of reach at the conceptual level.

OntoWeaver addresses this problem by means of the site view ontology which provides fine-grained modelling support for user interfaces and navigation structures of data-intensive web sites. In particular, the site view ontology addresses the specification of atomic user interface elements, generic composite user interface elements, as well as typical user interface elements. It realizes a composition mechanism and allows web developers to express complex user interfaces according to their own requirements.

Support for the design of the presentation layer

With the partial exception of OntoWebber, none of the current approaches mentioned earlier have provide high level support for the design of this layer. OntoWebber provides *a set of style elements* (e.g., font and colour) for specifying presentation styles for site view elements and *a set of layout primitives* (e.g., grid layout and flow layout) for specifying the layout of cards in web pages. However, the layout primitives can only be used to describe a fixed number of typical layouts.

OntoWeaver addresses this problem by providing an explicit presentation ontology to abstract the presentation styles and layouts of site view elements. In particular, the presentation ontology provides comprehensive constructs to allow the specification of complex layouts at the conceptual level.

Support for the design of the customization layer

As described in chapter 2, current web modelling approaches can only provide limited customization support for their target web sites. This is mainly caused by the lack of expressive meta-models for describing the target web site. OntoWeaver addresses this problem by means of the site ontologies. All aspects of web sites can be represented declaratively and are thus available for customization. Hence, the

scope of customization is not limited in OntoWeaver as it does in other approaches. Furthermore, as will be described in chapter 6, OntoWeaver separates the specification of customization from other aspects of the target web site and provides specific support for the specification of customization requirements.

5.5.2 Related work on user interface modelling

Substantial efforts have been spend in user interface modelling [Szekely et al., 1995] [Puerta, 1997] [Szekely, 1996] [da Silva, 2000], which try to reduce the amount of code that programmers need to produce when creating a user interface. The User Interface Markup Language (UIML) [Abrams et al., 1999] and the eXtensible Interface Markup Language (XIML) [Puerta & Eisenstein, 2002] are recent proposals which address the problem of authoring user interfaces for multiple platforms. These two languages are declarative, appliance-independent, and generic. However, they are very different from OntoWeaver. They do not target data-driven applications, and thus do not separate domain data, application logic, and user interfaces.

The XML User Interface Language (XUL)[3] is an approach proposed for describing user interfaces for web applications. It models graphic user interfaces into user interface widgets, such as *window*, *box*, *menu*, *tool bar*, *checkbox*, *button*, *scrollbar*, etc. While XUL provides more choices of widgets than OntoWeaver for rendering site view elements of web pages, XUL remains at a low level as it requires low-level programming in order to assemble graphic user interface widgets together as a meaningful user interface.

5.5.3 Related work on presentation modelling

The Cascading Style Sheets (CSS) and the eXtensible Stylesheet Language (XSL) are close approaches to our work on presentation modelling. Both of them provide

---

[3] The XML User Interface Language (XUL), available from: http://xulplanet.com/tutorials/xultu/ (Accessed 24 June 2005).

comprehensive means to support the specification of presentation instructions. While they are domain independent, they have their own specific targets. CSS targets HTML presentation objects, such as tables and hyperlinks. Hence it is appliance dependent and thus remains at a lower level than the OntoWeaver presentation ontology. XSL on the other hand focuses on the presentation of the source data stored in the specified XML document. It is in the same level with the presentation ontology. However, it only concerns the presentation instructions of those site view elements which present data objects retrieved from the source data. Other site view elements, such as elements for data acquisition and querying, are beyond its consideration. Hence, XSL is very different from our presentation ontology.

5.5.4 Related work on using ontologies to drive the generation of web sites

The use of ontologies to drive the generation of software tools has been demonstrated in the research fields of knowledge acquisition and semantic web portals, where a number of tools have been developed which use domain ontologies to drive the generation of knowledge acquisition tools (e.g., Protégé series [Eriksson et al., 1994] [Grosso et al., 1999] and Knote [Motta et al., 2000]) and semantic web portals (e.g., SEAL [Stojanovic et al., 2001], ODESeW [Corcho et al., 2003], and KAON [Volz et al., 2003]). These tools are very different from OntoWeaver, as their primary goal is to generate software tools from domain ontologies. They do not apply ontologies to model their target applications, thus lack conceptual level support for the design and development.

OntoWebber is an ontology-based web modelling approach, which shares a similar goal with OntoWeaver. It models web sites by means of a site ontology and thus provide high level support for the design and development of web sites. However, as studied in chapter 2, the OntoWebber approach can only provide limited support for the creation of complex user interfaces, due to the lack of fine-grained modelling support. OntoWebber also suffers from limited customization support.

Apart from the conceptual web modelling approaches OntoWebber and Hera, XWMF [Klapsing et al., 2001] and RSS [Beged-Dov et al., 2000] also use RDF-

based languages to describe web sites, like OntoWeaver does. XWMF aims to create a machine-understandable web site through exploiting RDF to model web application and its content. It provides a generic web engineering schemata as the basic vocabulary to model a web application. XWMF does not provide explicit site models, and is thus quite different from OntoWeaver.

RSS, which stands for RDF (or Rich) Site Summary, is a lightweight metadata description and syndication format. It provides a vocabulary to describe a "channel" consisting of URL-retrievable items. Each item consists of a title, link, and brief description. It models web sites in a very simple way.

5.5.5 Related work on reference models

In the research area of hypertext design and development, a number of models have already been developed to describe architectures of hypertext and hypermedia systems. The Dexter Reference Model [Halasz & Schewartz, 1994] is one of the main approaches. It divides a hypertext application into three layers: a within-component layer which covers the domain content, a storage layer which describes navigation structures, and a runtime layer which describes mechanisms supporting user interactions. This three-layer architecture has been extended in other models, such as the Munich Reference Model [Koch, 2001] and the Adaptive Hypermedia Application Model (AHAM) [De Bra et al., 1999b] by adding an adaptive layer for describing adaptive hypertext and adaptive hypermedia systems. These models however exclusively focus on the presentation of static hypertext, which can be defined at design time. They cannot be used to specify data-intensive web sites, where data content is typically stored in the back-end databases.

5.6 Conclusions

In this chapter we have presented one major component of the OntoWeaver approach, the site ontologies, which overcomes the problems associated with current approach, including i) limited support for the creation of complex site views, ii)

limited support for the specification of presentation instructions, and iii) limited support for customization.

The site view ontology addresses the first issue by providing fine-grained modelling support for user interfaces and navigation structures of the target web site. On the one hand, it provides comprehensive support for the specification of navigation structures. In particular, the site view ontology addresses static navigation patterns where no additional information is involved in navigation as well as dynamic navigation patterns where contextual information flow is required. On the other hand, unlike current approaches which only address typical user interface elements of web pages, the site view ontology also addresses atomic user interface elements (e.g., input fields and command elements) which can not be further decomposed into sub elements, and composite user interface elements (e.g., web pages and generic components) which are typically composed of a number of sub elements. Thus, the site view ontology realizes a composition mechanism and allows web developers to express complex user interfaces. At the same time, the site view ontology covers all site view elements found in data-intensive web sites. It supports the specification of static user interface elements whose content is defined at design time, dynamic user interface elements whose content is retrieved from back-end data sources at run time, and interactive user interface elements which allow end users to type in information and invoke the associated services.

The presentation ontology addresses the second open issue by providing high level support for the specification of layouts and presentation styles for user interface elements. It allows web developers expressing complex layouts and presentation styles at the conceptual level. Web developers no longer need to encode the specification into web page implementations, like they have to in other approaches.

The site view ontology and the presentation ontology allow the target web site to be represented declaratively. In particular, all user interface elements and their presentation instructions are described declaratively and as a result available for customization. The customization framework, another component of the OntoWeaver approach, exploits this advantage and provides comprehensive

customization support for the target web site at design as well as run time. This will be described in the following chapter.

Apart from comprehensive support for customization, the site ontologies can also benefit the design and development of web sites by supporting i) automatic site specification generation which produces default site specifications by means of using the domain ontology to instantiate the site view ontology, ii) semi-automatic site maintenance which supports the maintenance of conceptual links between site specifications and the underlying ontology, and iii) web site design critiquing which make use of a set of rules to assess the web site design result by reasoning upon the declarative site specifications and makes correction or improvement recommendations. We will describe these benefits in more detail in chapter 7.

As indicated in chapter 4, OntoWeaver focuses exclusively on data presentation, like other approaches do. The development of specific functionalities is still achieved through low-level programming. Web service technology [W3C, 2000b] [W3C, 2001a] enables access to remote content and application functionalities, independently of specific implementations or data formats. We believe there is a strong need for integrating web service technology into high level web site design frameworks to facilitate the specification of data-intensive web sites which not only allow access to the back-end data sources but also allow access to remote web services. In chapter 8, we will describe our work on the integration of (semantic) web services into OntoWeaver. The resulting architecture, OntoWeaver-S, supports rapid prototyping of web service centred web applications, customizable for different user profiles.

Chapter 6: Design of customized web sites

In chapter 5 we have described one major component of the OntoWeaver approach, which is the site ontologies. In this chapter we focus on another major component, the customization framework, which addresses the limitation of current web modelling approaches on customization support. We begin by briefly describing the research background of customization design of web sites. We then present an overview of the OntoWeaver approach to customization design and describe the major components of the OntoWeaver customization framework. Thereafter, we present a concrete example to illustrate how the OntoWeaver customization framework facilitates the design of customized web sites. In particular, we present a number of examples with respect to user group specification customization and rule-based customization. We also investigate the support that the OntoWeaver customization framework provides for the specification of data access permissions for user groups and individuals. Finally, we describe the related work and draw conclusions.

6.1 Introduction

As described in chapter 3, customization design for web sites has been addressed in a number of research directions. A number of customization techniques have been developed which facilitate user modelling, user information acquisition, and adaptation. Furthermore, a number of authoring tools have been developed, which address the design and development of adaptive hypermedia applications [Brusilovsky, 1996] [Kobsa et al, 2001]. These tools typically employ a user model to describe user profiles, a domain model to describe domain applications, and an adaptive engine to present adaptive interfaces to users. However, as studied in chapter 3, they do not offer high level support either for the specification of adaptive rules or for the design of web applications.

In the community of conceptual web modelling, a number of approaches have explicitly addressed the customization design issue. Two methods have been developed. They include composition-level customization (e.g., the approaches used in WSDM and OO-H) which allows the creation of different site views for different user groups and individuals and derivation-level customization (e.g., the approaches used in UWE, HERA, WUML, and the extended OOHDM [Rossi et al., 2001]) which derives customized views for individuals. However, as clarified in chapter 2, these customization solutions can only provide limited customization support. Firstly, the compositional-level customization can not scale up. Secondly, the scope of the derivation-level customization support is limited, as not all aspects of web sites are available for customization, due to the lack of expressive meta-models for describing the target web site. Furthermore, no specific support is available for the specification of customization requirements in derivation-level customization.

OntoWeaver approaches the design of customized data-intensive web sites by means of a set of site ontologies and a customization framework. As already described in chapter 5, the site ontologies enable all aspects of data-intensive web sites (including navigation structures, user interfaces, and their presentation instructions) to be represented declaratively. The customization framework makes use of this advantage and employs i) a user ontology which captures features of the information about end users, ii) a customization rule model which allows the specification of customization rules at design time, and iii) a customization engine to enable comprehensive support for customization at run time.

6.2 An overview of the OntoWeaver customization framework

OntoWeaver strictly separates the domain data model, the site view model, and the presentation model. The modular approach *per se* already supports user group specified customization by allowing the creation of different site models (i.e. site view models and presentation models) for different user groups. However, the dynamic (i.e. run-time) customization requirements of individual users can not be supported simply by this modular architecture. The customization should take place

dynamically according to the contextual information of each user individual. To this purpose, OntoWeaver proposes a generic customization framework to support individualization of the target web applications.

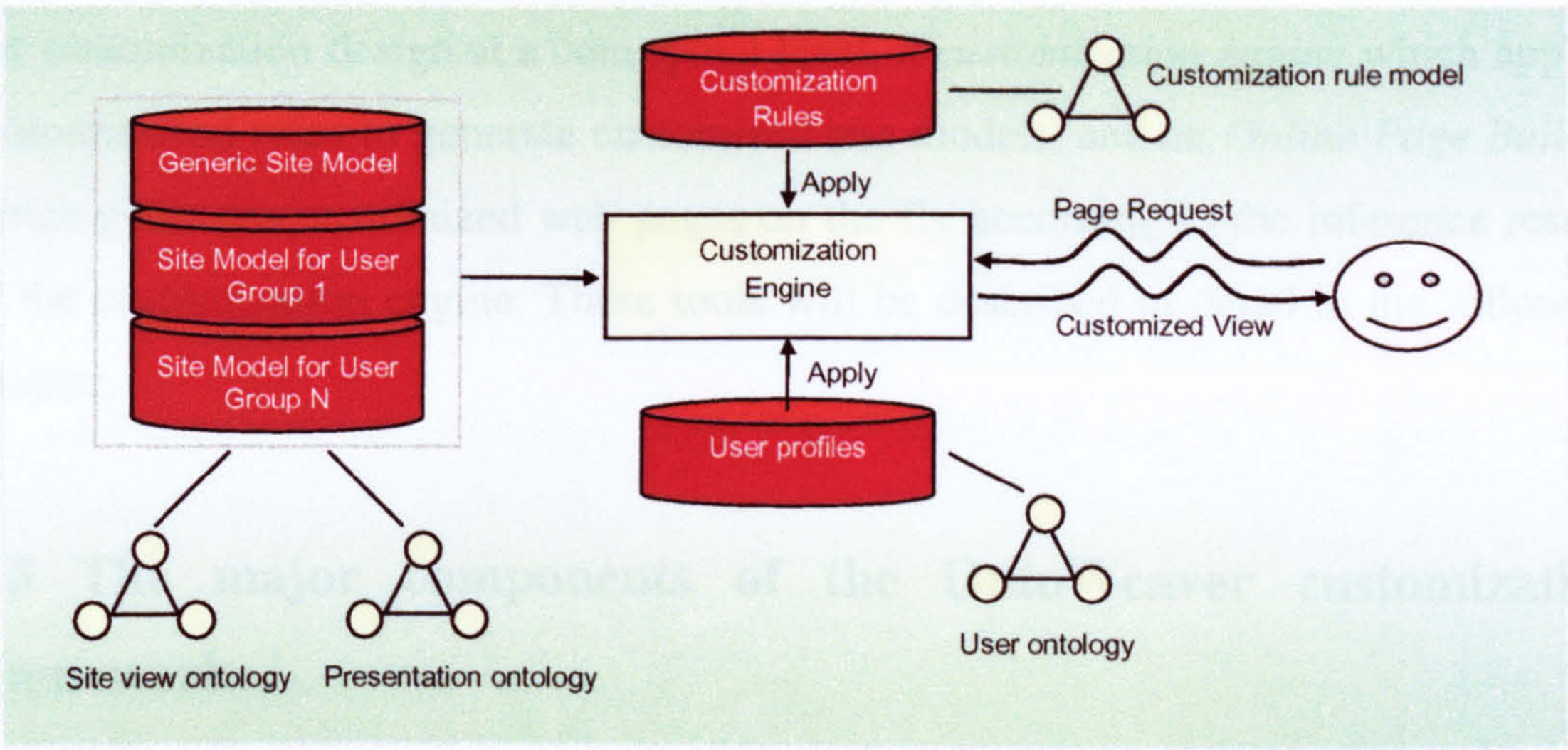


Figure 6.1 The OntoWeaver customization framework

Figure 6.1 shows an overview of the OntoWeaver customization framework which relies on *customization rules* to describe when and how to perform certain customization actions, *user profiles* to capture user information, and a set of *user group specific site models* to describe the target web site. The framework references site view elements specified in site models by means of their uniform resource identifiers, thus realizing the separation of customization specification from other aspects of the target web site. *A customization engine* is employed, which applies rules to reason upon user group specific site specifications to derive customized views for individuals according to facts stored in their profiles. These major components will be described in detail in the following section.

A typical design process for customized web sites in OntoWeaver proceeds by iterating the following two major activities: designing a general web site for general users and specifying customization requirements. Customization design further involves i) specifying a user ontology which describes information about end users, ii) creating different site views and presentations for different user roles, and iii) specifying customization rules for the individualization.

As described in chapter 4, OntoWeaver provides a suite of tools to allow developers to carry out these design activities at the conceptual level. In particular with respect to customization design and customization support, The OntoWeaver tool suite provides *a Site Customizer* which allows developers or web administrators to carry out customization design at a conceptual level, *a customization engine* which applies customization rules to generate customized site models, and *an Online Page Builder* which generates customized web pages on the fly according to the inference results of the customization engine. These tools will be described in detail in the following chapter.

6.3 The major components of the OntoWeaver customization framework

As already shown in figure 6.1, the major components of the OntoWeaver customization framework comprise the user ontology which supports the representation of user profiles, the customization rule model which allows the specification of customization rules, and the customization engine which performs customization at run time. In this section, we describe these components.

6.3.1 User ontology

A user ontology provides a vocabulary to represent information about end users. In most cases, user ontologies are domain dependent. In other words, the user ontology should be defined in a way that exactly reflects the key features of end users in a specific domain. OntoWeaver provides a generic user ontology to describe general information about end users. The user ontology contains two class entities: the class *User*, which describes user information in terms of *hasUserID*, *hasPassword*, *hasUserGroup*, *hasDevice*, and *hasInterest*, and the class *UserGroup*, which relies on slots *hasSiteViewURL* and *hasSitePresentationURL* to specify the user group specific site view model and the presentation model of the target web site. This ontology provides a basic support mechanism to allow end users to log into the target web application and to assign particular site views and presentations according to the

roles that end users belong to. This basic user ontology can be easily extended to abstract user information in the context of the problem domain.

The user ontology plays an important role in supporting customization. It makes possible for customization conditions to be defined on the basis of the user information at the conceptual level. Its instantiations, user profiles, are used to evaluate conditions of customization rules and decide whether the corresponding customization rules are to be fired or not. Each customization action takes place only when the dynamic context of end users meets a certain condition. For example, device-dependent customization takes place when the user device meets certain conditions.

6.3.2 The customization rule model

Customization rules define when and how to perform certain customization actions in terms of conditions and actions. The condition part describes a condition that has to be satisfied for the associated customization actions to take place. The action part describes the adaptation actions, e.g. adding/hiding/modifying site view elements, or setting presentation styles or layouts for them. To provide specific support for the construction of customization rules, OntoWeaver proposes a customization rule model. Figure 6.2 shows an overview of this rule model, whose major constructs are described in table 6.1.

Table 6.1 The major constructs of the customization rule model

Construct	**Slots**	**Description**
CustomizationRule	• hasCustomizationCondition • hasCustomizationAction	Describing customization rules
CustomizationCondition	• hasCondition	Describing conditions that should be satisfied for certain actions to be performed.
CustomizationAction	• hasSiteEntityURI • hasCustomizationActionType • hasSiteEntityType • hasParentSiteEntityURI • hasModification	Expressing how to perform adaptation actions. The slot *hasSiteEntityURI* indicates the site view element that the adaptation works on. The slot *hasCustomizationActionType* specifies the customization type (e.g., creating a new site view component, modifying the specified aspect of the associated site view element, hiding the specified site view element, or setting presentation instructions). The slot *hasSiteEntityType* specifies the type of the site view element in terms of the site view ontology classes. The slot h*asParentSiteEntityURI* expresses the site view element that holds the working site view element. Both of the latter slots are used when a new site view component is created in customization.

Modification	• hasSlotEntity • hasNewValue	Describing the details of how to customize the specified site view element. The slot *hasSlotEntity* expresses the adaptation aspect of the specified site view element in terms of the site ontologies. The slot *hasNewValue* specifies the new value.
Condition	• hasClassEntity • hasSlotEntity • hasRelationOperator • hasSpecifiedValue • hasLogicOperator	Formalizing conditions.

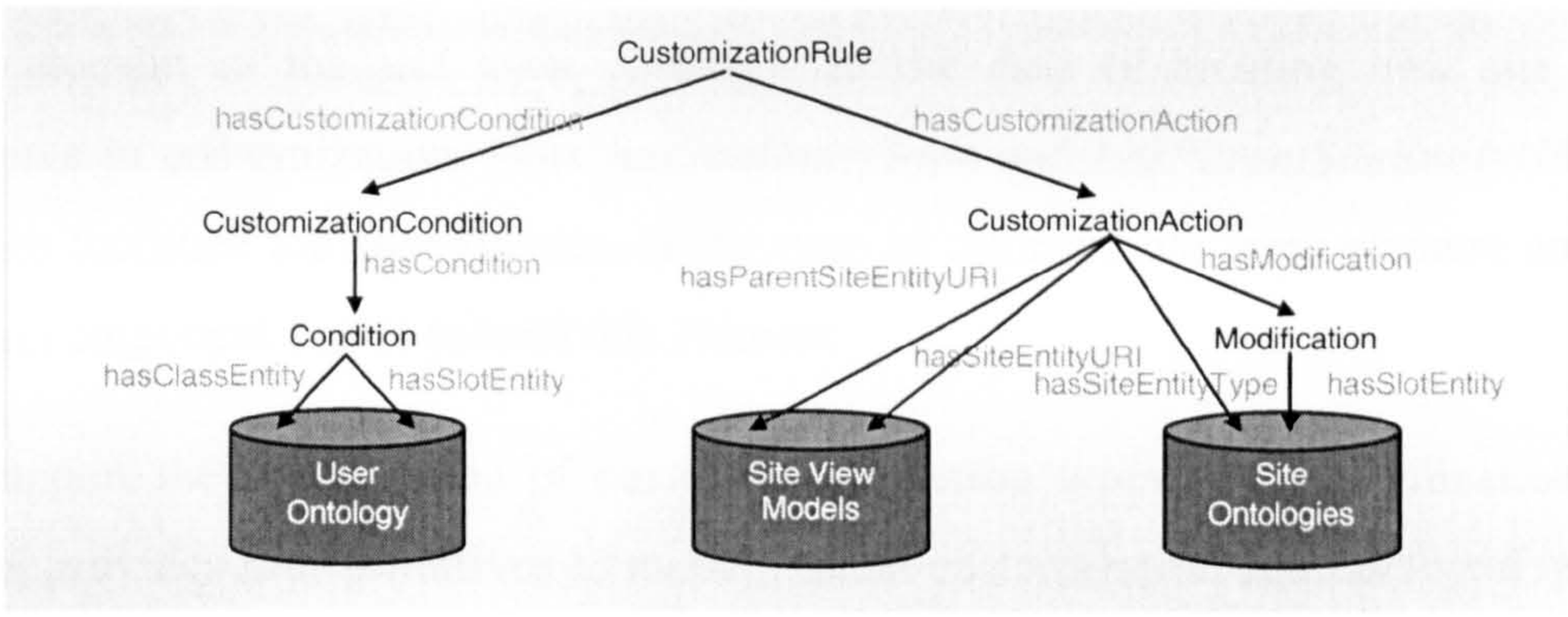


Figure 6.2 An overview of the customization rule model

A customization condition can be atomic, which comprises only one condition, or composite, which is composed of a list of conditions by means of logical operators such as *AND*, *OR*, or *NOT*. Each condition is formulated by means of the construct *Condition*, which relies on slots *hasClassEntity* and *hasSlotEntity* to specify the user data which are going to be evaluated, and slots *hasRelationOperator* and *hasSpecifiedValue* to define the way to evaluate the condition, and the slot *hasLogicOperator* to specify how to connect this condition with the next one. As described in chapter 5, the site view ontology provides a number of primitives to support the specification of relation operators and logic operators. Examples include *EQUAL*, *NOT-EQUAL*, *GREATER-THAN*, *NO-LESS-THAN*, *LESS-THAN*, *NO-GREATER-THAN*, *NOT*, *AND*, and *OR*.

The specification of a customization action typically comprises three components: the slot *hasSiteEntityURI* which indicates the site view element that the action works on, the slot *hasCustomizationActionType* which specifies the customization type (e.g., creating a new site view component, modifying the specified aspect of the associated site view element, hiding the specified site view element, or setting presentation instructions), and the slot *hasModification* which expresses

customization details. Specifically, the slot *hasModification* relies on the class *Modification* to describe the customization details in terms of *hasSlotEntity* - *hasNewValue* pairs. The slot *hasSlotEntity* indicates the aspect that customization intends to work on in terms of the site ontologies, while the slot *hasNewValue* specifies the customized value. For example, when customizing content of a site view element, the slot *hasSlotEntity* refers to a certain slot entity defined for the site view element in the site view ontology. In the case of creating new site view elements in customization, slots *hasSiteEntityType* and *hasParentSiteEntityURI* are used to facilitate the specification of the type of the new site view element and the parent component that is to hold this element.

To support the specification of customization action types, the customization rule model provides four primitives to model typical customization actions found in web sites. They are i) the primitive *MODIFY*, which indicates that the customization action will modify certain aspects of the specified site view element, ii) the primitive *NEW*, which specifies that a new site view element will be created in the customization, iii) the primitive *HIDE*, which expresses that the customization action will hide the specified site view element, and iv) the primitive *PRESENTATION*, which indicates that the customization action works on the presentation styles or layouts of the specified site view element.

To illustrate how this rule model facilitates the specification of customization requirements, we now investigate an example which customizes the project web page of the KMi web portal (as illustrated in chapter 5) for presenting only those projects that match the interest of end users. This web page relies on the project data component which is an instance of the construct *DataComponent* to present instances of the class *Project*. As described in chapter 5, constraints can be specified on a data component to filter instances which are to appear in terms of the slot *hasInstanceConstraint* and the construct *InstanceConstraint*. Hence, this customization can be achieved by using the interests of an end user as a constraint to filter those irrelevant papers.

This customization requirement can be expressed by one customization rule. The condition part specifies that the value of the slot *hasInsterest* of an end user should not be empty. The action part comprises two actions: i) creating an instance constraint which uses the interest of end users as constraints of the slot *addresses-research-area* to filter instances of the class *Project*, and ii) associating the newly generated instance constraint with the project data component. The following fragment of RDF code[1] defines this customization rule.

```
<rdf:Description rdf:about="…/customization-rule1" >
  <rdf:type rdf:resource="&sco;CustomizationRule" />
  <sco:hasCustomizationCondition rdf:resource="…/customization-rule1/condition"/>
  <sco:hasCustomizationAction rdf:resource="…/customization-rule1/action1" />
  <sco:hasCustomizationAction rdf:resource="…/customization-rule1/action2" />
</rdf:Description>

<rdf:Description rdf:about="…/customization-rule1/condition" >
  …
  <sco:hasCondition rdf:resource="…/condition1" />
</rdf:Description>

<rdf:Description rdf:about="…/condition1" >
  <rdf:type rdf:resource="&sco;Condition" />
  <sco:hasClassEntity rdf:resource="&uo;User" />
  <sco:hasSlotEntity rdf:resource="&uo;hasInterest" />
  <sco:hasRelationOperator rdf:resource="&svo;NOT-EQUAL" />
  <sco:hasSpecifiedValue rdf:resource="&svo;NULL" />
</rdf:Description>

 <!-- The first customization action: creating a new instance constraint -->
<rdf:Description rdf:about="…/customization-rule1/action1" >
  <rdf:type rdf:resource="&sco;CustomizationAction" />
  <sco:hasCustomizationActionType rdf:Resource="&sco;NEW" />
  <sco:hasSiteEntityURI>…/project/datacomponent/instanceconstraint
  </sco:hasSiteEntityURI>
  <sco:hasParentSiteEntityURI>…/project/datacomponent</sco:hasParentSiteEntityURI>
  <sco:hasSiteEntityType rdf:resource="&svo;InstanceConstraint" />
  <sco:hasModification>
     …
     <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedClassEntity" />
     <sco:hasNewValue rdf:resource="&do;Project"/>
     …
  </sco:hasModification>

  <sco:hasModification>
     …
     <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedSlotEntity" />
     <sco:hasNewValue>"&do;addresses-research-area"</sco:hasNewValue>
     …
  </sco:hasModification>

  <sco:hasModification>
     …
     <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedRelation" />
     <sco:hasNewValue rdf:resource="&svo;EQUAL" />
     …
  </sco:hasModification>

  <sco:hasModification>
     …
     <sco:hasSlotEntity rdf:resource="&svo;hasConstrainedValue" />
     <sco:hasNewValue>"&uo;User.hasInterest"</sco:hasNewValue>
     …
  </sco:hasModification>
</rdf:Description>
```

---

[1] The prefix 'sco' refers to the namespace of the customization rule model: xmlns:sco=http://kmi.open.ac.uk/people/yuangui/sitecustomizationontology#. The prefix 'uo' refers to the namespace of the user ontology.

```
<!-- the second customization action: associating the instance constraint with the project data
component -->
<rdf:Description rdf:about="…/customization-rule1/action2" >
  <rdf:type rdf:resource="&sco;CustomizationAction" />
  <sco:hasSiteEntityURI>…project/datacomponent </sco:hasSiteEntityURI>
  <sco:hasCustomizationActionType rdf:Resource="&scc;MODIFY" />
  <sco:hasModification>
     …
     <sco:hasSlotEntity rdf:resource="&svo;hasInstanceConstraint" />
     <sco:hasNewValue rdf:resource="…/project/datacomponent/instanceconstraint"/>
     …
  </sco:hasModification>
</rdf:Description>
```

In this rule example, the first customization action defines how to create an instance constraint by specifying i) the type of the newly created site view element in terms of *hasSiteEntityType*, ii) where to put the newly created element in terms of *hasParentSiteEntityURI*, and iii) how to create the site view element in terms of the construct *Modification*, in which specifies the details of the constraint. The second customization action indicates i) the working site view element, which is the project data component and ii) how to customize the component, which is to set value for the slot *hasInstanceConstraint*.

6.3.3 The declarative site model

Only limited support for customizing web applications can be provided if the specifications of web applications are hidden from the customization engine. For example, as studied in chapter 3, the WUML approach can only provide limited customization support for web applications even though it provides comprehensive support for the capture of customization context for web sites, as the specifications of web applications are not available. It relies on web developers to define adaptation hooks in web site specifications to achieve customization support. The OntoWeaver approach overcomes this problem by considering customization modelling together with web application modelling. As a result, the entire site model is available to the customization engine. The declarative site model serves as the foundation to facilitate customization:

- The declarative site specifications and their underlying ontologies make it possible to specify customization actions precisely at the conceptual level. OntoWeaver employs RDF models to represent specifications. As a result, each item possesses a Uniform Resource Identifier (URI) to identify itself. Moreover, its underlying

ontology definition allows the precise specification of the modifications that the action will perform. For example, if we want to customize a particular output element to a particular user context, we need to be able to identify this output item using its URI, and specify the modification information about its content, e.g. changing the output style, and associating/removing hyperlinks with/from it, according to the definition of the construct *Output* in the site view ontology.

- The declarative site model provides facts to be reasoned upon to generate customized site models.

6.3.4 The customization engine

The customization engine applies the relevant customization rules to perform inferences upon the declarative site specifications and thus produces personalized web sites for individual users. It employs the Jess[2] inference engine to perform the inferences. Jess is a Java expert system shell, which provides a scripting language to define facts and rules and offers an inference engine to reason about the facts.

To enable the deployment of the Jess inference engine, we have developed an RDF-to-Jess compiler, which compiles OntoWeaver ontologies, site specifications, user profiles and customization rules into Jess templates, facts and rules. Specifically, the site ontologies and the user ontology are compiled into Jess templates. The site specifications (i.e. the manipulated site ontologies) and user profiles are compiled into Jess facts. Regarding the customization rules, the condition part is compiled into the left-hand-side part of the Jess rules, and the customization action part is converted into the right-hand-side. A rule in Jess looks like: LHS => RHS, where LHS is a conjunction of conditions and RHS is a conjunction of actions. The following Jess rules are translated from the declarative specification of the rule example discussed above. The first Jess rule creates a new instance constraint when the condition is satisfied. The second rule associates the instance constraint with the project data component.

---

[2] http://herzberg.ca.sandia.gov/jess/ (Accessed 24 June 2005)

```
(defrule customization_rule0
  (user (hasInsterest ?x))
  (test(<> ?x ""))
  =>
(assert(InstanceConstraint(siteentityuri
"…/project/datacomponent/instanceconstraint")
       (hasConstrainedClassEntity "&do;Project")
       (hasConstrainedSlotEntity "&do;addresses-research-area")
       (hasConstrainedRelation "&svo;equal")
       (hasConstrainedValue ?x)))
```

```
(defrule customization_rule1
  (user (hasInsterest ?x))
  (test(<> ?x ""))
  ?se <- (DataComponent (siteentityuri "…/project/datacomponent"))
  =>
   (modify ?se (hasIntanceConstraint "…/project/datacomponent/instanceconstraint")))
```

The customization process starts when an end user logs into the OntoWeaver-generated web site and makes a page request. The tool *Online Page Builder* receives a page request from the end user, gets user-group specified site model (i.e. site view URL and presentation URL), invokes the customization engine to apply the specified rules to perform inferences upon the specified site model, gets the inference result from the customization engine, and builds the customized web page and delivers it to the end user.

Regarding customization rules controlling, the customization framework groups customization actions into one rule when their customization conditions are the same. Thus, when one specific condition is satisfied, only one rule will be fired. However, there are still situations where more than one rule could be applied. In such circumstances, the customization engine relies on the physical order of customization rules stored to decide their priority. This strategy is far from comprehensive. In future work, we will investigate several possibilities to acquire rules priorities, e.g., automatic or semi-automatic discovery or manual acquisition.

6.4 Example application: building a conference paper review system

In this section, we illustrate the capability of the OntoWeaver approach to the design and development of customized web applications by building a conference paper review system. The goal of the conference paper review system is to provide support for the management of conferences during their life cycle. In particular, the conference paper review system offers support for the process of conference set up,

paper submission, paper review, and paper selection. In the life-cycle of a conference, there are typically four types of user roles involved in: *chairs*, *authors*, *program committees, and reviewers*. Conference chairs set up conferences by submitting information (e.g., conference descriptions and calls) to the paper review system. Authors submit papers to the conferences. The conference chairs and program committees assign papers for reviewers. Reviewers review the specified papers and send review results to the conferences. Chairs notify authors about the acceptances of their papers.

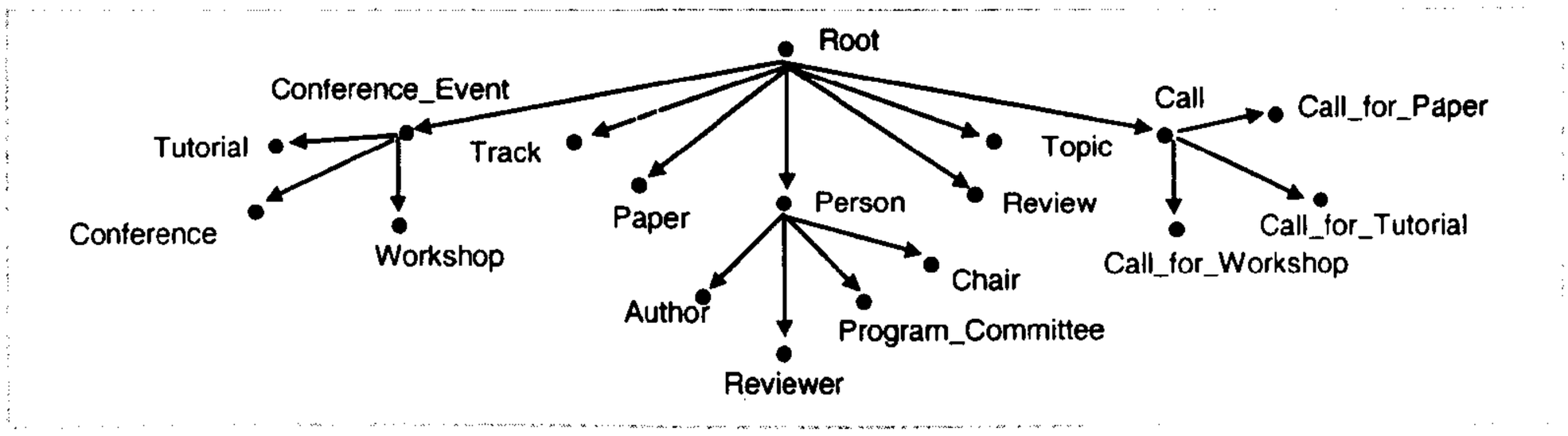


Figure 6.3 The domain ontology of the conference paper review system

6.4.1 The domain ontology

The design process in OntoWeaver starts with the definition of the domain ontology which abstracts data structures of the problem domain. As shown in figure 6.3, the domain ontology of the paper review system comprises the following class entities: the class *Conference_Event* which abstracts events such as conferences, workshops and tutorials; the class *Call* which models the calls for papers, workshops, and so on; the classes *Topic* and *Track* which describe the topics and tracks of the conference; the class *Paper* which abstracts the submitted papers; the class *Review* which expresses the review information about papers, and the class *Person* which models people involved in the paper review system. An instantiation of this domain ontology forms a specific conference.

6.4.2 A generic view of the paper review system

In this section, we specify a view for general end users, who can browse the conference information and submit papers to the conference. This general view comprises a number of page nodes, which are organized in three categories:

- The *Calls* category comprises *a call-for-paper page*, *a call-for-workshop page,* and *a call-for-tutorial page*. Through browsing information in this category, end users can get information about the specified conference;
- The *Submission* category comprises pages that allow end users to submit papers, tutorials, and workshop proposals;
- The category of *about the conferences* consists of web pages presenting important dates, program committees, and contact information.

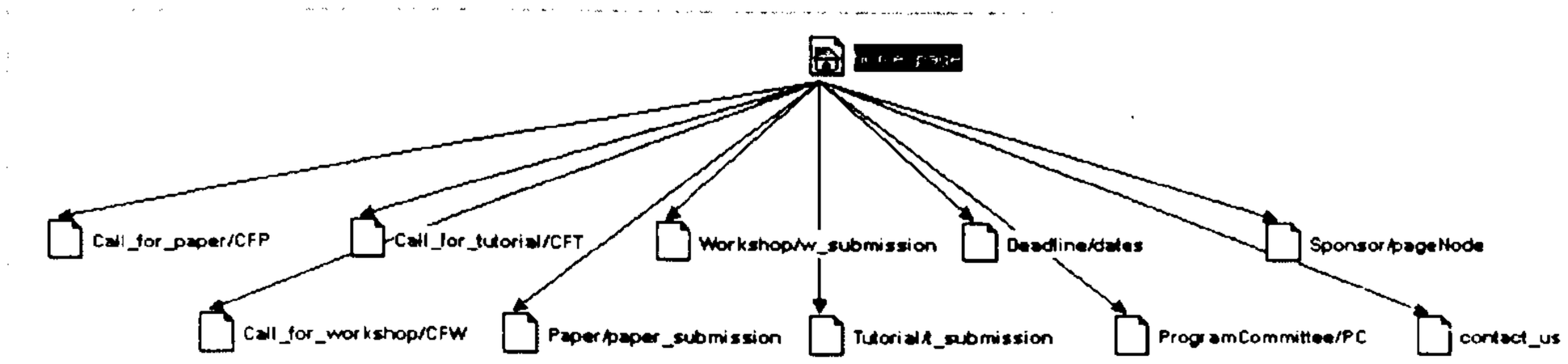


Figure 6.4 The simplified navigational structure of the general paper review system

Figure 6.4 shows the simplified navigational structure of the general paper review system, which allows the navigation from the index page to other pages. Each web page is connected with other pages through the same navigation pattern. Please note that OntoWeaver provides a tool called *Site Designer* to support web developers achieving the task of specifying navigational structures, composing user interface and defining presentation styles and layouts for web pages. The OntoWeaver Site Designer will be described in chapter 7. Figure 6.5 shows the index page of the conference paper review system, which is designed by means of the OntoWeaver Site Designer.

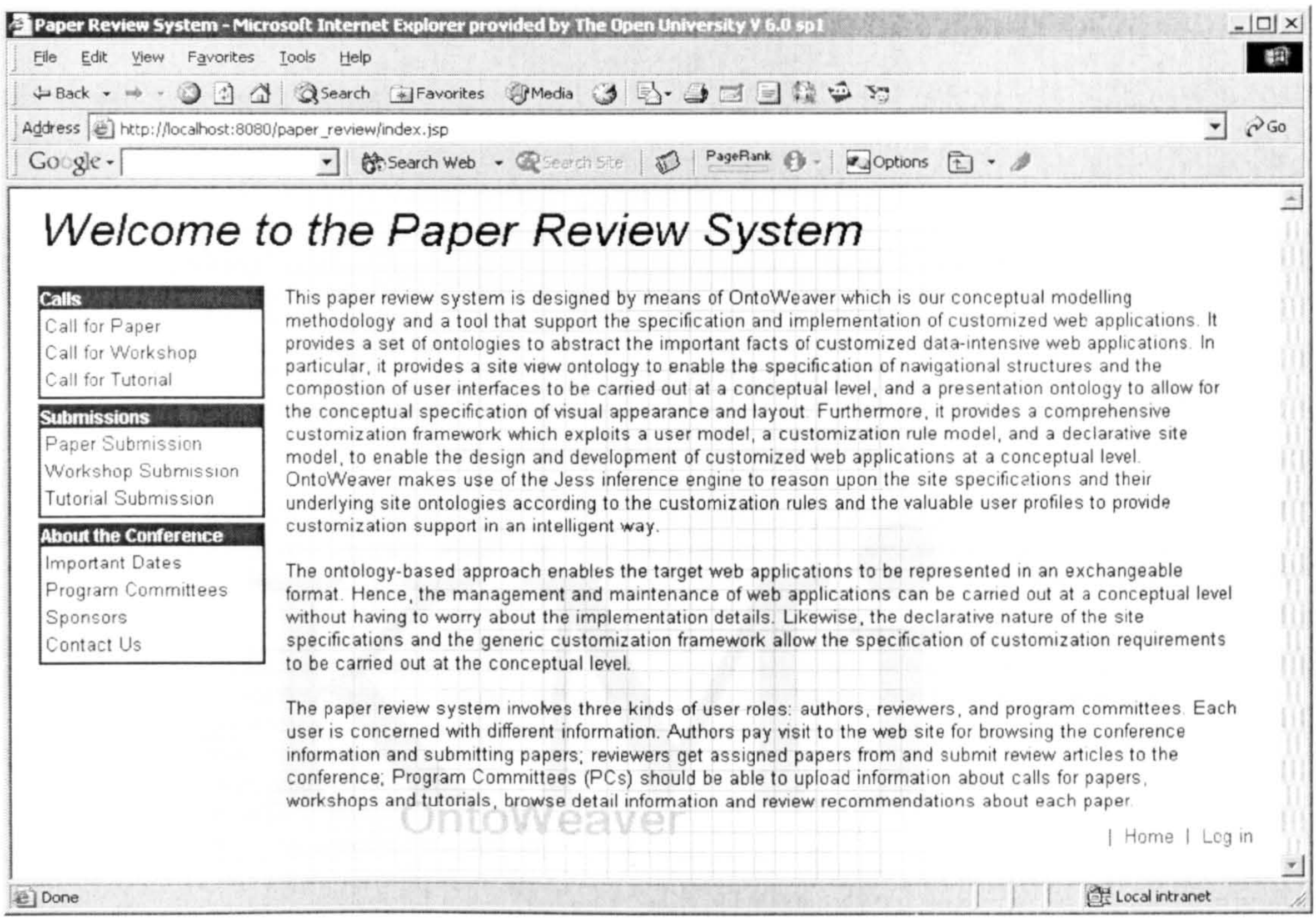


Figure 6.5 A screenshot of the home page of the conference paper review system

6.4.3 User group specific customization design

Two kinds of customization are involved in the conference paper review system. They are user group specific customization which presents different site views to different user roles (i.e. user groups), and rule-based customization which takes the contextual information of end users into consideration and personalizes web pages for individual users. Although we discuss these two kinds of customization separately, it should be noted that both types are combined together to support customization. In particular, the rule-based customization in OntoWeaver is based upon user group specific customization, as it reasons upon user group specific site models. In this section we describe the user group specific customization design in the paper review system. The rule-based customization design will be described in the following section.

As mentioned earlier, the paper review system involves four types of user roles: *authors*, *reviewers*, *program committees*, *and chairs*. Each user group has different

tasks and thus requires a particular site view. Table 6.2 lists the requirements of site views for each user group:

Table 6.2 The requirements of site views for user groups in the conference paper review system

User Group	Site View Requirements	Particular Web Pages Required to Be Created
Chairs	• Conference set up. • PC member registrations. • Review assignment. • Acceptance notification.	• Web pages for the submission of the call for papers, the call for workshops and the call for tutorials. • A web page for the creation of program committees. • A web page for the assignment of papers for the program committees and reviewers. • A web page for the browsing of detailed information of papers.
Program committees	• Assigning papers for Reviewers. • Submitting review results to the conference.	• A web page for the creation of paper reviewers. • A web page for the assignment of papers for paper reviewers. • A web page for the presentation of the assigned papers for program committees. • A web page for the submission of review results.
Reviewers	• getting assigned papers; • Submitting review results to the conference;	• A web page for the presentation of the assigned papers. • A web page for the submission of review results.
Authors	• Paper Submission; • Workshop Submission; • Tutorial Submission; • Browsing submitted information;	

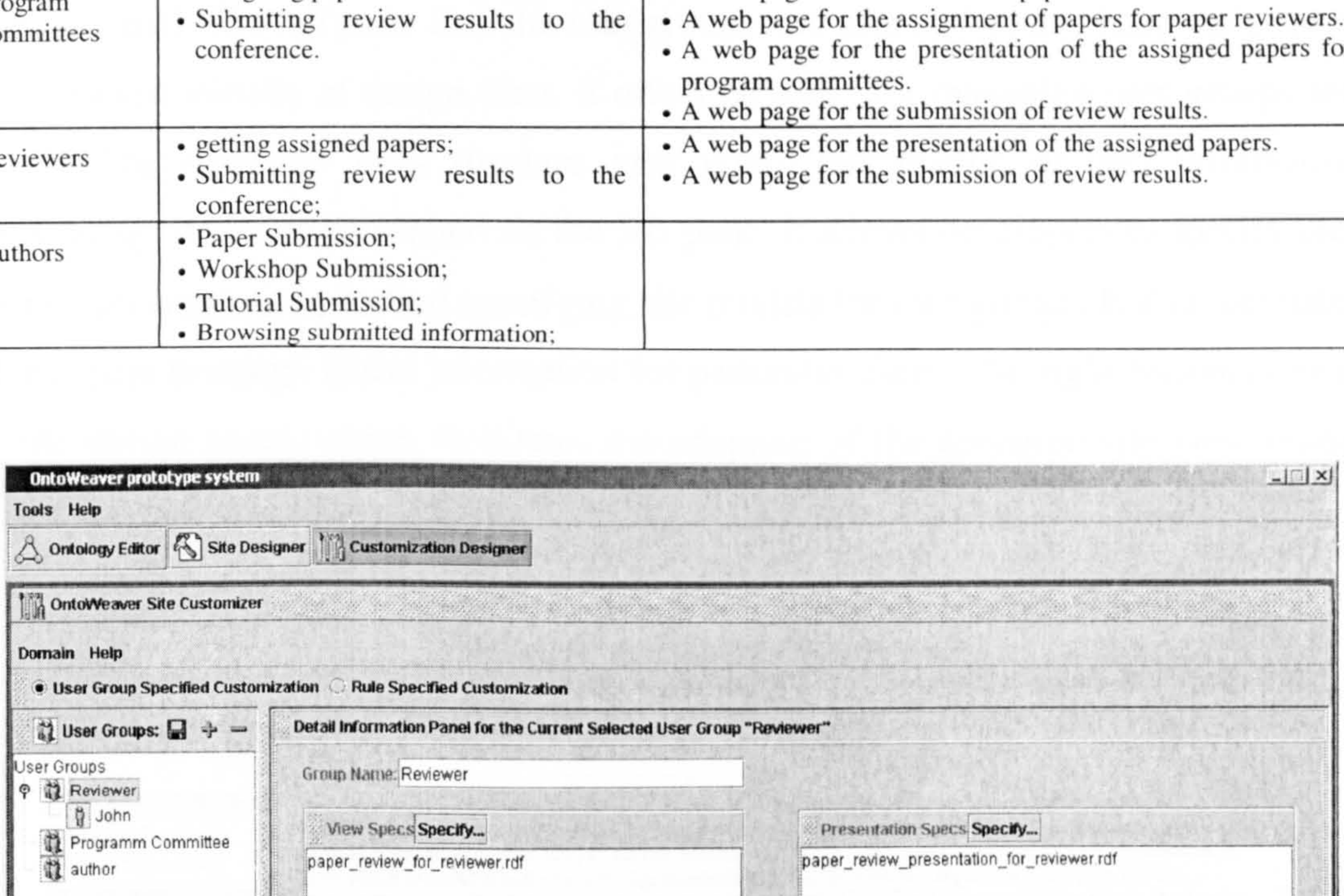


Figure 6.6 A screenshot of the user group customization design pane, which is contained in the OntoWeaver Site Customizer.

Like other approaches, the OntoWeaver strategy to user group specific customization is creating different site models for different user groups. Web developers can use the generic site model as the basis to create user group oriented site models. This task can be easily achieved by means of the OntoWeaver *Site Customizer*, which provides *a user group customization design pane* to facilitate the creation and management of user groups, and the derivation of site models for user groups. In fact, the process of editing user group specified site models remains the same with editing generic site models. Figure 6.6 shows a screenshot of the OntoWeaver user group customization design panel. The *left pane* lists the user groups and users who have been assigned to user groups initially at design time. It offers facilities for managing user groups and users. The *right top pane* displays user group information or user information depending on the item selected on the left pane. It allows developers to specify user group information, including specifying site models for user groups. It also facilitates developers to assign initial information for particular users. The *right bottom pane* is a site design panel, which facilitates the adapting of the specified site view model towards the specified user group.

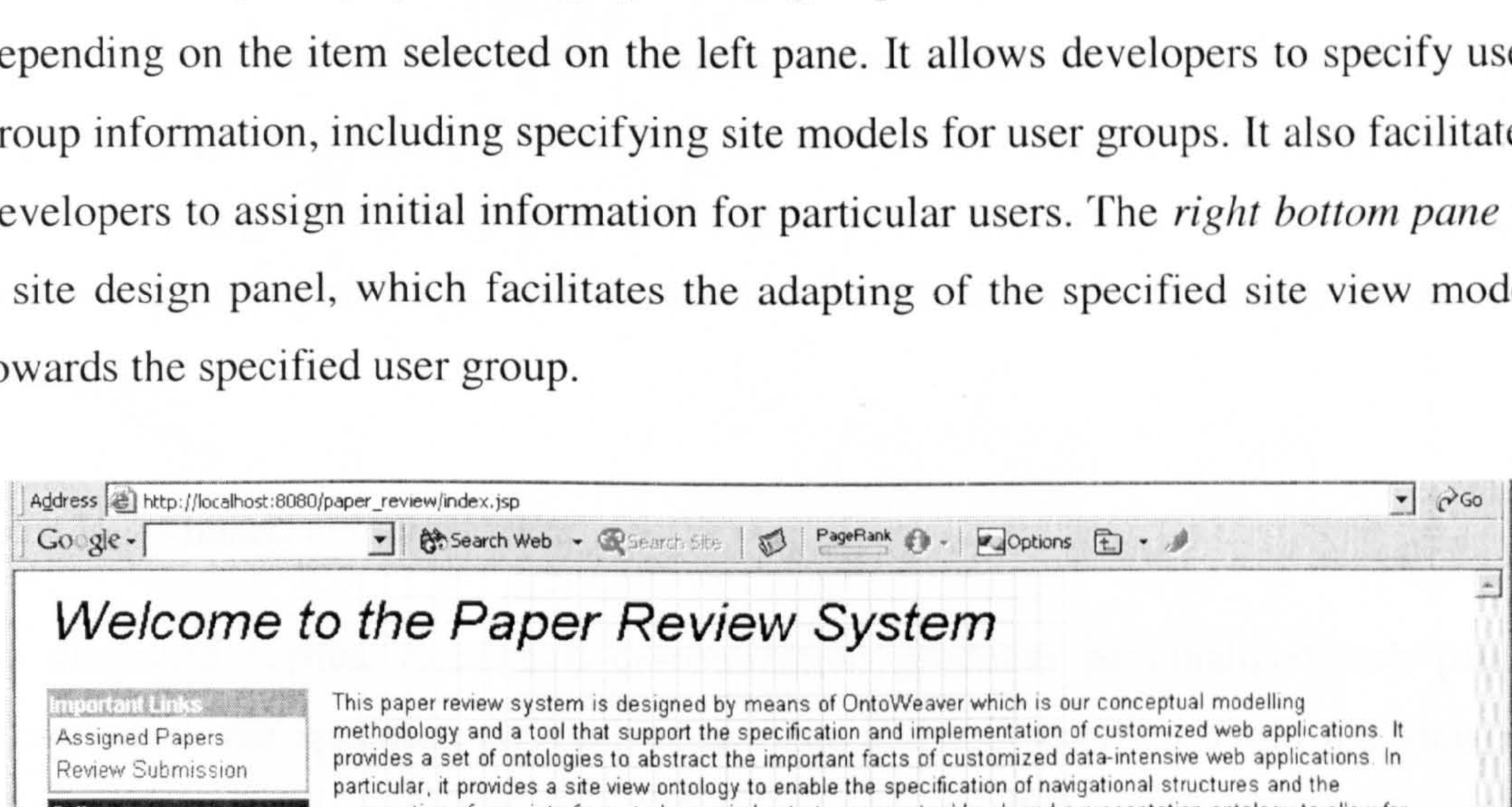


Figure 6.7 A screenshot of the index page of the paper review system for reviewers

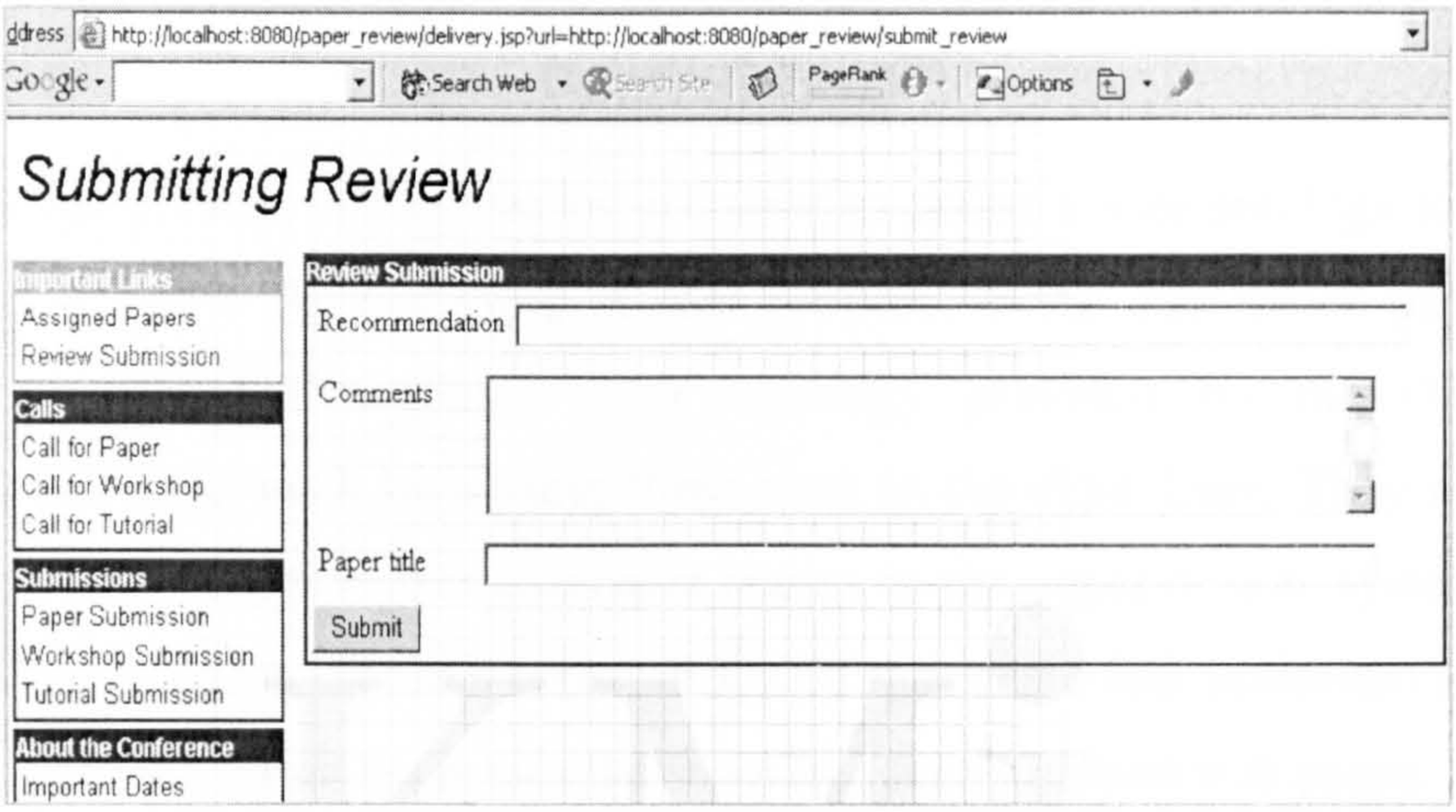


Figure 6.8 A screenshot of the review submission web

Figure 6.7 and figure 6.8 show the customized index page and the review submission page for paper reviewers. Two extra links are created for paper reviewers, which allow them to get assigned papers and submit review comments. At the same time, a new web page is generated, which contains a form to allow the submission of review comments.

6.4.4 Rule-based customization

As discussed earlier, rule-based customization produces personalized web pages according to the specified customization rules and user profiles. From the view of what may be customized in the information space that end users perceive, as discussed in chapter 3, there are mainly two types of customization actions: link customization, which works on the link topology, and content customization, which tailors contents of web pages according to individual's requirements. From the presentation instructions point of view, there is another customization, called presentation customization, which customizes presentation styles or layouts for individual users. Although there are mainly these three customization actions, the personalization context could be any, e.g. time dependent, location dependent, device dependent, user interest dependent, and so on. The so-called dynamic customization personalizes the links or content of web pages according to the contextual information of end users which is typically determined at run time. In this section,

we will use a number of examples to illustrate the OntoWeaver approach to dynamic customization.

Before we can present rule examples, we need to define a user ontology to provide a vocabulary to allow the recording of user profiles. In the conference paper review system, we extend the generic user ontology provided by the OntoWeaver customization framework by adding three slots to the class *User*. They are the slot *time* which describes time that end users access to the paper review system, the slot *interestedTopic* which captures research topics that users feel interested in, and the slot *favoriatePage* which expresses the most frequently visited web pages.

Please note that the tool *Site Customizer* provided by the OntoWeaver tool suite provides *a rule-based customization design pane* (which will be described in chapter 7) to support the specification and management of customization rules. Thus, there is no need for web developers to deal with the raw data of rule specifications.

Link customization

Link customization involves improving the original navigation spaces by reducing or improving the relationships between page nodes. It provides links that are more relevant to individual users. This customization strategy is popular in E-Commence web sites, which recommends items based on the client buying history or on the clients who have similar ratings and opinions [Rossi et al., 2001]. For example, Amazon.com[3] recommends new products to individual users according to their interests. This type of customization involves generating links based on the underlying domain data content with appropriate constraints. For instance, in the Amazon.com scenario, the new products links are generated on the instances of new products using user's interests as filters to constrain the new products content. In the context of the conference paper review system, this customization strategy can be used to improve the usability of the system, for example: i) recommending related

[3] http://www.amazon.com (accessed 18 October 2004)

papers to program committees, ii) presenting favourite links for end users, and iii) hiding submission links after the deadlines of the submissions have been passed. In the following, we will illustrate how we use the OntoWeaver approach to achieve the goal of these customizations. Please note that there is no need for web developers to deal with the raw data of customization rules, the tool *Site Customizer* offers visual facilities to support the specification. This tool will be discussed in the following chapter.

Example 1: Recommending related papers. In this example, the related papers are recommended upon the interest of program committees which is recorded in their user profiles in terms of *interestedTopic*. Figure 6.9 shows an informal description of this customization rule. This rule creates a data component which presents the titles of the papers whose topics match with the *interestedTopic* of program committees. The condition part specifies that only when the user is one of the members of program committees and his or her interested topic is not empty the customization can take place. The action part comprises three sub-actions to create the required data component: i) creating a data component which presents titles of papers, ii) creating an instance constraint which filters related paper instances for the data component, and iii) creating a link item which is associated with the dynamic output element that presents paper titles to allow the navigation to the web page presenting detailed information about the related paper. As in section 6.3.2 we have illustrated how to specify customization actions which create new elements for the site view model, we will not describe the code here.

Customization rule name: paper_recommendation_rule
Condition: the role of an end user is "PC member" and the interestedTopic is not empty
Action: (1) create a data component presenting titles of the instances of the class *Paper*
(2) create an instance constraint which filter instances of the class Paper
(3) create a link item for the dynamic output element of presenting paper titles.

Figure 6.9 An informal description of the paper recommendation rule

Customization rule name: favourite_link_rule
Condition: the favourite page of an end user is not null
Action: (1) create a favourite link component
(2) create an output item which is associated with a link leading to each favourite page

Figure 6.10 An informal description of the favourite link generation rule

Example 2: recommending favourite links. The provision of favourite links in web pages can provide shortcuts for users browsing their favourite web pages. In this rule example, the customization condition part describes that a user should have favourite web pages to enable the presentation of these web pages, the action part describes creating a component in the navigation component and creating a list of output elements, which are associated link items leading to the favourite web pages. Figure 6.10 shows an informal definition of this customization rule.

Example 3: Time-dependent customization. The view of the paper review system should be time dependent. For example, before the paper submission deadline, the paper submission link is valid and allows users to submit papers to the system. After the deadline, the link should be removed from the system. Likewise, the notification time is a crucial time for publishing information about accepted papers. In these circumstances, customization happens when time changes and meets particular conditions. The following RDF fragment specifies the customization rule which hides the paper submission link item when the paper submission deadline has passed.

```
<rdf:Description rdf:about="…/deadline-rule-for-paper" >
  <rdf:type rdf:resource="&sco;CustomizationRule" />
  <sco:hasCustomizationCondition rdf:resource="…/deadline-rule-for-paper/condition"/>
  <sco:hasCustomizationAction rdf:resource="…/deadline-rule-for-paper"/action" />
</rdf:Description>

<rdf:Description rdf:about="…/deadline-rule-for-paper/condition">
  …
  <sco:hasCondition rdf:resource="…/condition1" />
</rdf:Description>

<rdf:Description rdf:about="…/condition1" >
  <rdf:type rdf:resource="&sco;Condition" />
  <sco:hasClassEntity rdf:resource="&uo;User" />
  <sco:hasSlotEntity rdf:resource="&uo;time" />
  <sco:hasRelationOperator rdf:resource="&svo;GREATER-THAN" />
  <sco:hasSpecifiedValue rdf:resource="&do;Paper.deadline" />
</rdf:Description>

<rdf:Description rdf:about="…/deadline-rule-for-paper"/action" >
  <rdf:type rdf:resource="&sco;CustomizationAction" />
  <sco:hasSiteEntityURI>…/link-for-submit-papers </sco:hasSiteEntityURI>
  <sco:hasCustomizationActionType rdf:resource="&sco;HIDE" />
</rdf:Description>
```

Page content customization

The difference between content customization and link customization is subtle, as when links are customized page content can be seen customized as well. Nevertheless, here we are talking about personalizing general content, i.e. general information presented on web pages.

Example 1: Customizing domain data content. Domain data content customization involves customizing data content according to different requirements of end users. In the conference paper review system, program committees might only want to browse details about those papers that match their preferences. The web page relies on a data component element to present paper details. We can achieve this customization by using the preferences of an end user as a parameter to filter those irrelevant papers. The customization condition is exactly the same as the condition for presenting related papers for program committees. The modification part of the customization action specifies an instance constraint for the specified data component. Suppose John is a member of the program committee; his preference for papers is about customization. When he logs onto the conference paper review system as a member of program committees, he gets papers which are related to the topic of customization. This rule is very similar with the rule example which has been illustrated in section 6.3.2. The customization action part comprises two actions: i) creating a new instance constraint and ii) associating the new instance constraint with the data component which presents domain data content of the specified class entity.

Example 2: Device-dependent customization. Device-dependent customization computes content of web pages according to the devices that an end user employs to access the web site. We can specify rules according to the different requirements of different devices. For example, when the screen size of the device that an end user employs to access the target web site is smaller than a certain size, the content of site view elements that present large pieces of text should be briefed. Figure 6.11 shows an informal specification of a customization rule which briefs the introduction text of the index page of the conference paper review system.

Customization rule name: device_dependent_rule_example
Condition: the screen size of the device that an end user users is smaller than a certain size
Action: tailoring the content of the output element which presents introduction for the index page

Figure 6.11 An example of a device-dependent customization rule

Presentation customization

Presentation customization involves in customizing the visual appearance or layout of site view elements according to the requirements or preferences of the end users. It defines particular visual appearances or layouts for the specified site view elements (i.e. navigational elements and user interface elements). The following specification example defines a customization action, which customizes the fore colour of the specified site view element.

```
<rdf:Description rdf:about="presentation_action_example" >
   <rdf:type rdf:resource="&sco;CustomizationAction"/>
   <sco:hasCustomizationActionType rdf:resource="&sco;PRESENTATION">
   <sco:hasSiteEntityURI>…/indexpage/introductionComponent</sco:hasSiteEntityURI>
   <sco:hasModification>
      …
      <sco:hasSlotEntity rdf:Resource="&spo;hasForeColor"/>
      <sco:hasNewValue>"#668866"</sco:hasNewValue>
      …
  </sco:hasModification>
</rdf:Description>
```

6.4.5 Data access permission as customization

Now we illustrate the support that the customization framework provides for the specification of data access (i.e. browsing and writing) restrictions for user groups and individuals. While data browsing restrictions restrain end users from browsing sensitive information, e.g., particular instances of a class concept or values of particular slots, writing restrictions specify whether an end user is granted permission to insert new facts to the underlying database of the target web site or not.

As described in chapter 5, OntoWeaver relies on the constructs *DataComponent* and *KAComponent* to describe site views for data browsing and knowledge acquisition. Thus, data access permission can be realized by creating or deriving different views for data components and knowledge acquisition components for user groups or individuals. In particular, data access restrictions can be specified by means of user group specific customization when they target groups of end users or by rule-based customization when the restrictions work on the basis of individuals.

In the conference paper review system, confidential review comments are only available to conference chairs. Authors do not have access to such data. This can be

achieved by means of creating different views for the data component which present the instances of the class *Review* for conference chairs and authors. Specifically, the data component comprises all the elements needed for presenting values of all slots of the class for conference chairs, while in the site view of authors the element which presents values for the slot *confidential_comments* is not included in the data component.

More complex permission works on the basis of individual users. Suppose John is a member of program committees. In addition to the general data access rights specified for program committees, John is also granted permission to view confidential comments from reviewers. Such data access permission for individuals can be specified by a customization rule in OntoWeaver. In this example, the condition part specifies that the end user should be John. The action part indicates that a new dynamic output element should be added to the review data component contained in the site view of program committees for presenting values for the slot *confidential_comments*.

6.5 Related work

In this section, we compare our work to those closely related customization approaches developed in different research directions.

6.5.1 Related work on conceptual web modelling for customization

WebML, OOHDM, and HERA are examples of web modelling approaches, which consider customization issues within the design phase of web applications. Like OntoWeaver, they support user group specified customization by allowing the definition of different site models for different user groups. Regarding support for run-time customization, the overall mechanisms of these approaches are very different from OntoWeaver. They annotate the site model and make use of the annotations to determine the visibility or the values of components. The problem of these approaches is that they do not offer high level support for the annotation. Moreover, given the limitations on site view modelling (as dicussed in chapter 2), the

customization scope of these approaches is limited to dynamic data content and partial web content.

WUML is an approach which exclusively focuses on customization modelling. It proposes a generic customization model to support the capture of user profiles and customization requirements. However, as the WUML approach does not support modelling of web applications, it requires web developers to slice the target web applications into a stable part, comprising context-independent structure, and a variable context-dependent part, which is the subject of the adaptations. Furthermore, it requires web applications to provide adaptation hooks to enable customization. Thus, it greatly limits the possibility for customization by forcing site developers to anticipate what may be customized (i.e. annotate the web site model by defining extra adaptation hooks). Table 6.3 describes the OntoWeaver approach and other related conceptual web modelling approaches on how they support user group specific and user individual specific customization.

Table 6.3 A comparison of the OntoWeaver customization approach to other related conceptual web modelling approaches on how they support user group specific and user individual specific customization

Approach	**User group specific customization**	**User specified customization**
OntoWeaver	• Creating different site views for different user groups. • Generating web pages on the fly according to the corresponding site models for end users.	• Creating rules for customization • Relying on the customization engine and the online page builder to generate individualized web pages.
WebML	• Creating different site views for user groups. • Translating different site views into different page templates. • Mapping users to groups • Mapping groups to different site views.	• Designing a derived domain model on the basis of the user-independent structural model. • Extending the generic site view model by defining contents of the derived concepts in terms of the profile data of individual users.
OOHDM	• Creating different navigational model and user interface model for different user groups.	• Adding additional descriptions to the components of the site view model for computing user dependent information for web pages.
HERA	Not explicit	• Adding conditions to the components (i.e. slices) of the site view model to compute the visibilities of the components. • The conditions are expressed using the user profile information.
WUML	Not explicit	• Customization requirements are specified in the format of rules in terms event/condition/action.

6.5.2 Related work on customization tools

As described in chapter 3, a number of customization tools have been developed, which aim at i) tailoring the applications' interactive behaviour to knowledge, skills,

tasks, and preferences of end users, ii) providing personalized recommendations or personalized assistant, or iii) customizing systems towards the access context indicated by the ubiquity of web applications. These customization tools focus on providing customization support for end users when they interact with web applications. They provide comprehensive support for user modelling and user information acquisition. OntoWeaver on the other hand concentrates on the specification of customized web applications. In particular, it provides an explicit rule model to support the specification of customization rules. At the same time, OntoWeaver assumes that user profiles are gathered by external tools and thus does not provide means to address the gathering of user information. Hence, OntoWeaver is very different from these customization tools. Nevertheless, the comprehensive techniques developed in these tools can be integrated into OntoWeaver, thus enabling more comprehensive customization support. This will be our future work.

6.6 Conclusions

In this chapter we have presented the OntoWeaver customization framework which addresses the limitations of current web modelling approaches on customization support. In particular, we have illustrated how the customization framework facilitates the design and development of customized web sites by building an example customized web application – the paper review system. We have also investigated the support that the OntoWeaver customization framework provides for the specification of data access permissions for user groups and individuals.

An important advantage of the OntoWeaver customization approach is that the entire site model is available to customization, and thus the subjects of customization are not limited. Another important advantage is that the OntoWeaver approach offers explicit high level support for customization design: customization rules can be specified at the conceptual level due to the fact that OntoWeaver exploits ontologies as the backbone to approach web site design and offers an explicit customization rule model to facilitate rule specification. Furthermore, the customization engine provides comprehensive customization support at run time for the target web site by applying

the specified rules to reason upon web site specifications according to the facts stored in user profiles.

The OntoWeaver customization approach offers the capability to provide meta-level customization support for the target web applications, as it supports web applications exploiting various techniques to collect user specified information, as long as user profiles are recorded as instances of the specified user ontology. OntoWeaver can make use of the user profiles to get contextual information from the end user, and exploit the customization engine to reason about site specifications to present customized web pages. Finally, as mentioned earlier, OntoWeaver provides a suite of tools to support the design and development of customized web applications which will be described in the following chapter.

One limitation of the OntoWeaver customization framework is that it does not provide comprehensive support for the handling of rules' priorities. As mentioned earlier, the customization engine makes use of two methods to address this issue. The first is grouping of customization actions with the same customization condition into one rule. Thus, whenever one specific condition is satisfied, only the corresponding rule will be fired. The second is making use of the physical order of customization rules stored to determine their priorities. These methods are far from comprehensive. In future work, we will investigate several possibilities to capture rules priorities, e.g., automatic or semi-automatic discovery or manual acquisition.

Chapter 7: The OntoWeaver tool suite

Building a data-intensive web site is a complex task which involves not only technical issues, but also organizational, managerial and artistic issues [Morville & Rosenfeld, 1998]. Different roles are thus involved in the design process. For example, domain experts design models that can abstract data of a specific problem domain. Information architects structure, organize, and label information. Site designers put the design into practice. Layout designers make decisions on presentation styles and layouts. Web administrators manage and maintain the web site. The OntoWeaver tool suite is built upon the OntoWeaver modelling approach.

In this chapter we explain how the OntoWeaver tool suite supports different roles involved in the design process to achieve their tasks. We begin by describing the facilities provided by the tool suite. We then illustrate how these facilities support different design roles by building an example data-intensive web site. Thereafter, we illustrate the benefits gained by using site ontologies to represent web site specifications, including automatic site specification generation and web site maintenance. Next, we compare the facilities provided by OntoWeaver to the facilities provided by other closely related conceptual web modelling approaches, with respect to the support for web site specification, customization design, web site design critiquing, and web site maintenance. Finally, we draw conclusions about the OntoWeaver tool suite.

7.1 The implementation of the OntoWeaver tool suite

As shown in figure 7.1, the OntoWeaver tool suite comprises the following major components:

- *A knowledge warehouse*, which hosts the ontologies, domain knowledge bases, site specifications, user profiles, and customization rules.

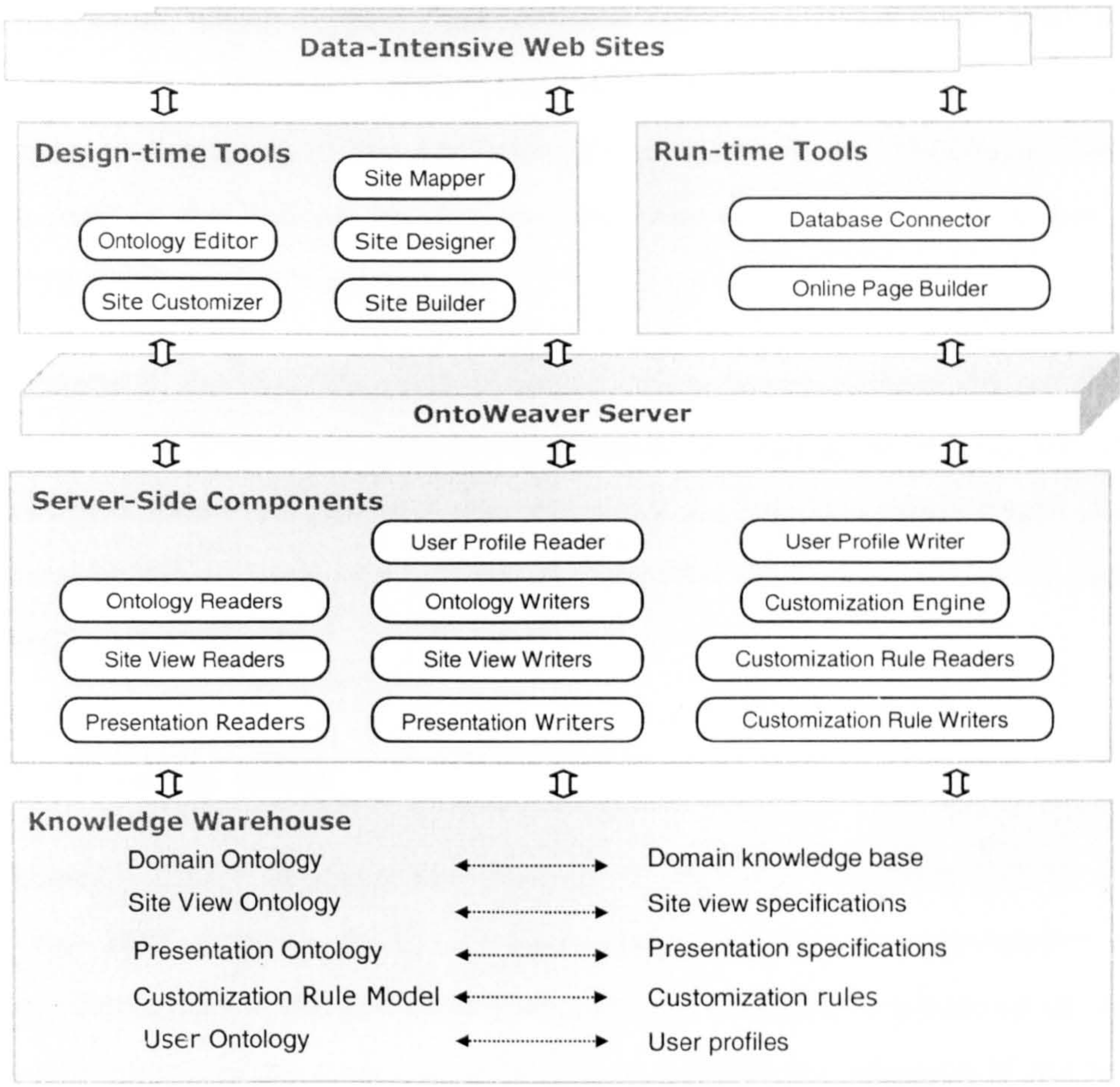


Figure 7.1 The architecture of the OntoWeaver tool suite

- *Server-side components*, which provide services for i) the design-time tools and the run-time tools to access and manipulate the ontologies and their data sets stored in the knowledge warehouse and ii) performing customization inferences at run time.
- *An OntoWeaver server*, which connects the OntoWeaver tools and the server-side components.
- *Design-time tools*, which provide a fast prototyping environment to support the design of data-intensive web sites. They comprise *an Ontology Editor*, which allows domain experts to create and edit the ontologies relevant to the web site, *a Site Designer*, which supports the design tasks needed for specifying a data-intensive web site, *a Site Mapper*, which supports automatic site generation and web site re-engineering, *a Site Builder*, which performs site design critiquing and compiles site specifications into web site implementations, and *a Site Customizer*, which supports the customization design.

- *Run-time tools*, which support data-intensive web sites at run time. They include the *Online Page Builder* which generates customized web pages on the fly according to the result of the customization engine and the *Database Connector* which realizes the built-in services to allow the access and manipulation of the underlying domain knowledge base.

The prototype of the OntoWeaver tool suite is implemented in Java. As mentioned in chapter 5, the site generation tools in OntoWeaver (including the Site Builder and the Online Page Builder) compile web site specifications into Java server pages (JSP). In the following sub-sections, we will briefly describe the functionalities of each tool contained in the tool suite.

7.1.1 The Ontology Editor

The Ontology Editor supports the creation of ontologies in both OCML [Motta, 1999] and RDF Schema [W3C, 2000a]. Figure 7.2 shows a screenshot of the Ontology Editor. While the left pane visualizes the hierarchy structure of the domain concepts by means of the Java tree technique and allows the selection of the working concept, the right pane provides a corresponding form which allows the specification of the working concept. The form can be either a class definition form or a property definition form, which depends on the type of the working concept. Furthermore, both forms support the specification of the association relations between class entities and property entities.

In the Ontology Editor, the creation of ontologies can be carried out either from scratch or from existing ontologies. The latter facility is realized by being able to read ontologies represented in textual documents located in local as well as in remote web servers. As shown in figure 7.1, the Ontology Editor relies on a set of server-side components to access and update ontologies. Specifically, *an OCML Reader* and *an RDF Schema Reader* are provided to read ontologies from the specified documents. At the same time, *an OCML Writer* and *an RDF Schema Writer* are provided to update ontologies.

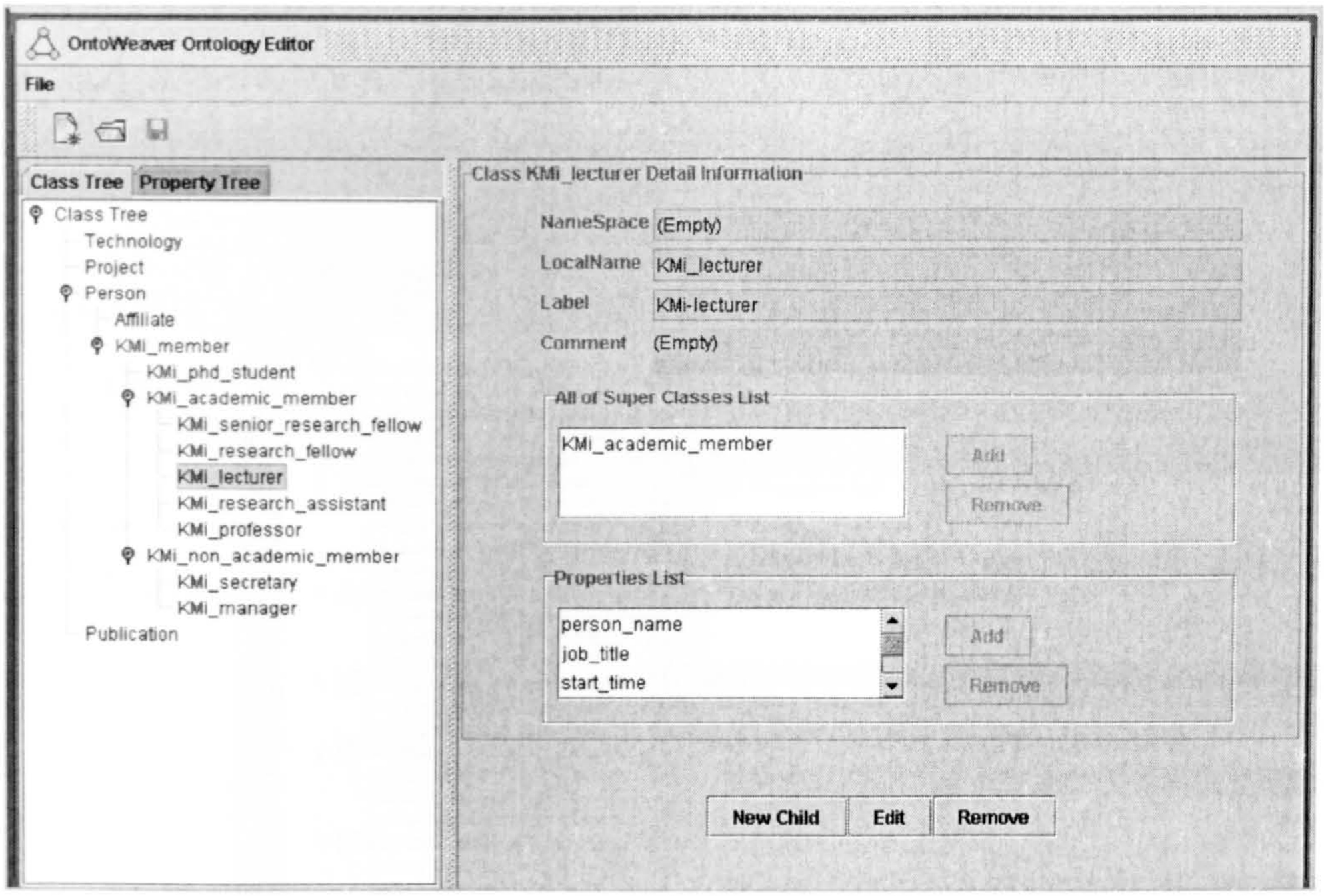


Figure 7.2 A screenshot of the Ontology Editor. The left pane visualizes the hierarchy structure of domain entities and allows the selection of the working concept. The right pane supports the specification of the specified working concept. In this screenshot, the working concept selected in the left pane is a class entity. The right pane thus presents a class definition form which allows the specification of this class. The form also supports the association of existing properties with the class in question.

7.1.2 The Site Designer

As mentioned earlier, the tool *Site Designer* supports the design tasks of specifying site structures, composing web pages, and defining visual appearances. It comprises *a Site Structure Designer*, *a Page Content Designer*, *a Presentation Designer*, and *a Layout Designer*. Figure 7.3 shows a screenshot of the Site Designer and illustrates its major components.

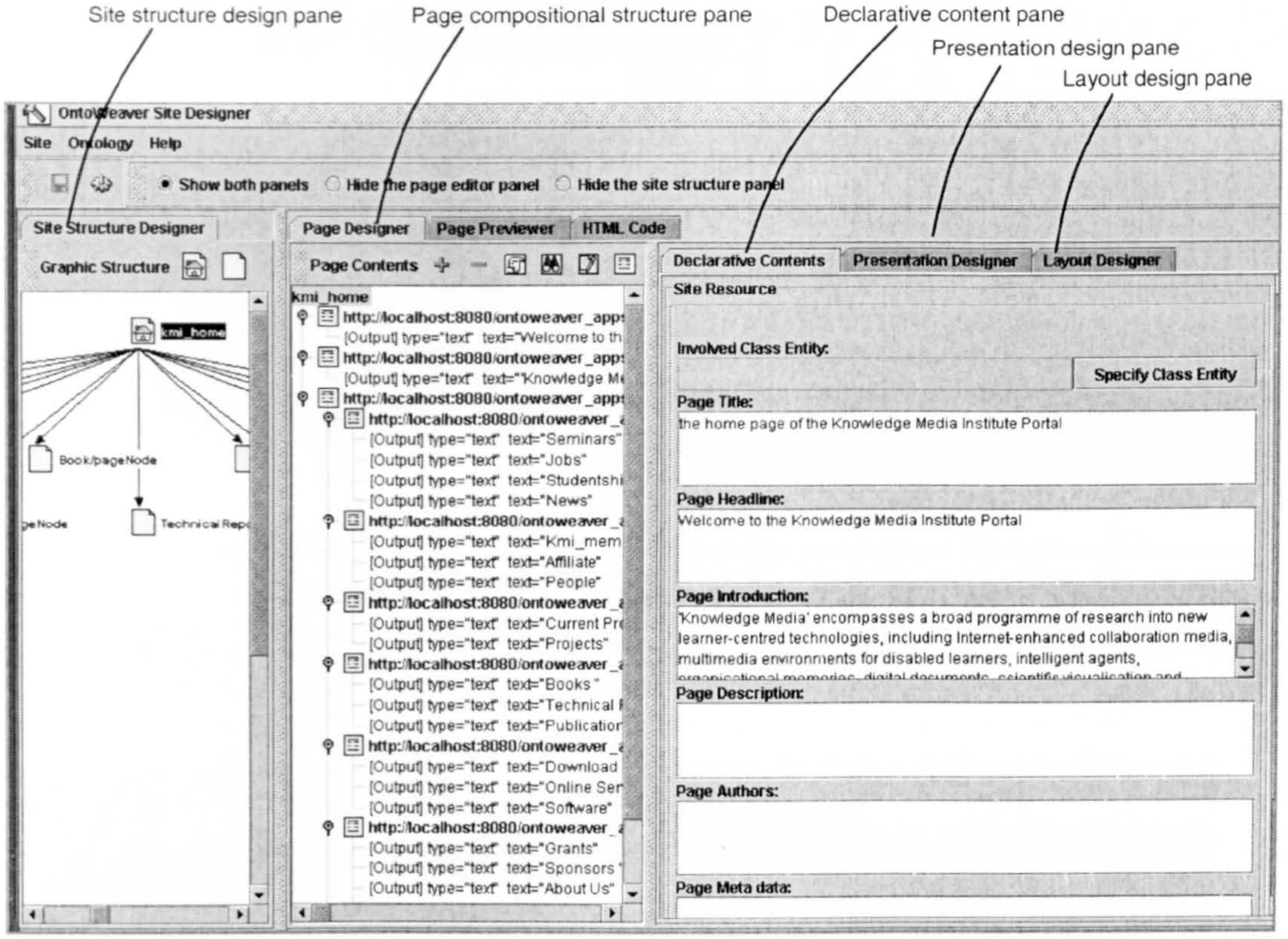
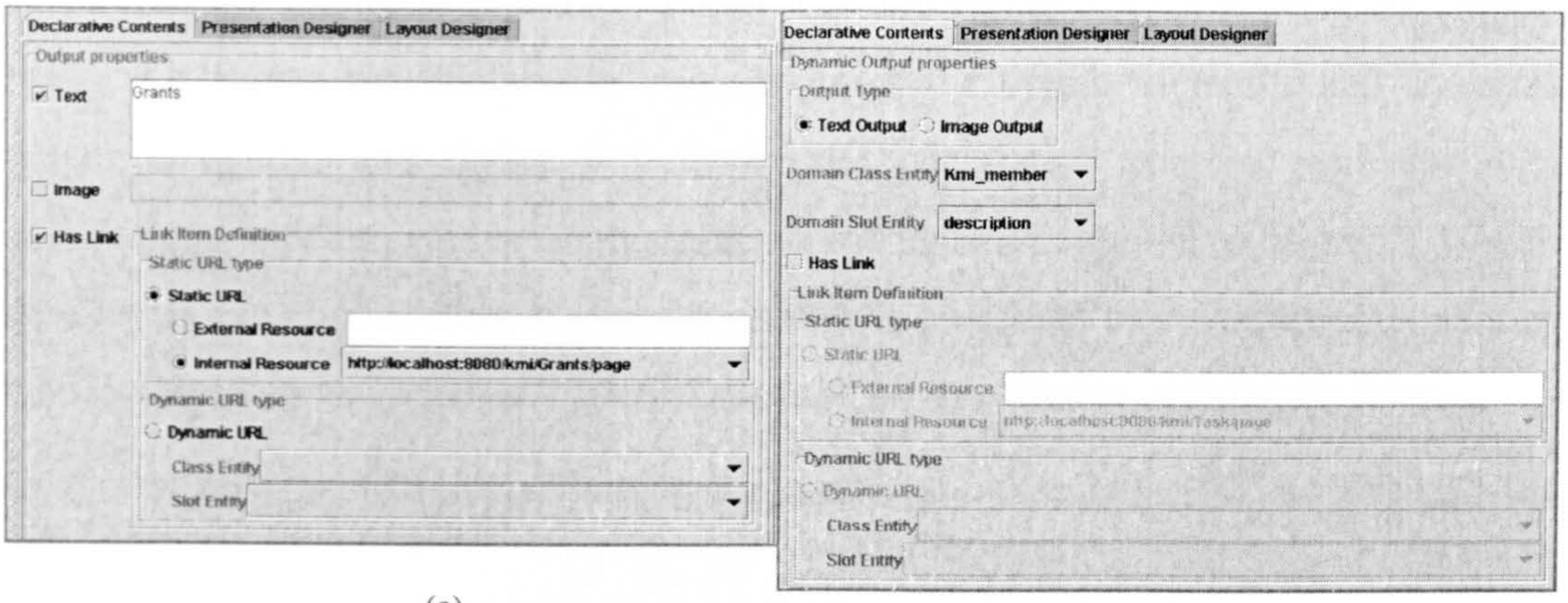


Figure 7.3 A screenshot of the Site Designer. The left pane is the *site structure designer pane*, which supports the creation and management of site structures and allows the selection of the working page node. The middle pane is the *page compositional structure pane*, which visualizes and manages the compositional structure of the specified page node. The pane also provides a preview facility and an implementation code viewing facility for page nodes. The right pane comprises three tools: *a declarative content pane*, which allows the editing of the declarative content of the selected user interface element, *a presentation pane*, which supports the specification of presentation styles for site view elements, and *a layout pane*, which allows the specification of the organization of the selected user interface elements.

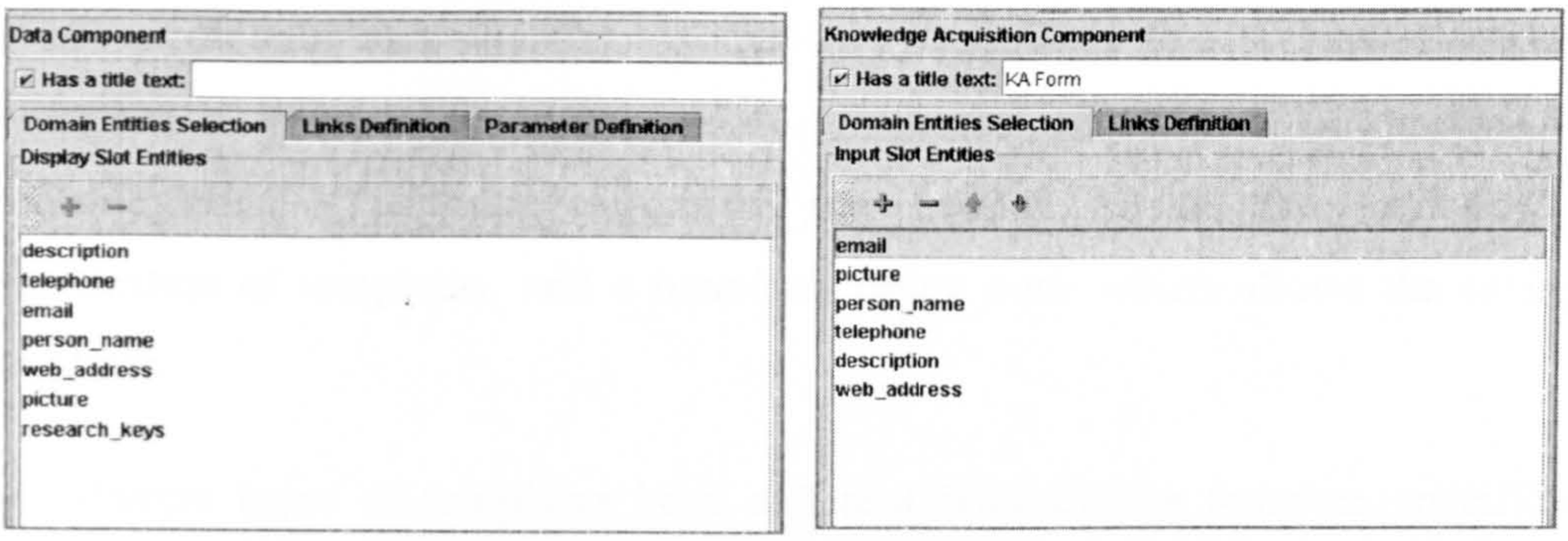
The Site Structure Designer

The Site Structure Designer supports the design of site structures. As illustrated in figure 7.3, the Site Structure Designer relies on *the site structure designer pane* to achieve its task. Site structures are visualized in a graphic layout: each web page is represented as a *node*; and each link is visualized as a *line* with a particular direction. The Site Structure Designer supports the management of page nodes and their link relations. Furthermore, it allows the selection of the working page node from the site structure.



(a) (b)

Figure 7.4 Screenshots of the declarative content panes for static output elements and dynamic output elements. Part (a) shows the declarative content pane for static output elements, through which web developers can specify the type and the content of the information which needs to be presented. Part (b) shows the content pane for dynamic output elements. Developers can specify the source of dynamic data content as well as information type.



(a) (b)

Figure 7.5 Screenshots of the declarative content panes for *data components* and *knowledge acquisition components*. Part (a) shows the content pane for data components. Part (b) shows the content pane for knowledge acquisition components. Web developers can choose slots whose values will be presented in the data component or whose value can be gathered from end users in the knowledge acquisition form for the associated class.

The Page Content Designer

The Page Content Designer allows the specification of detailed content for site view elements. As shown in the middle pane and the right pane of figure 7.3, the Page Content Designer relies on i) the *page compositional structure pane* to visualize and manage the compositional structures of web pages and ii) *the declarative content pane* to support the editing of the declarative content of each site view element contained in a page node. A number of *declarative content panes* are provided to allow the specification of different site view elements according to the constructs

defined in the site view ontology. Figure 7.4 and figure 7.5 show some examples: figure 7.4 shows the declarative content panes for output elements and dynamic output elements; and figure 7.5 shows the declarative content pane for data components and knowledge acquisition components. As shown in figure 7.5, web developers can choose slots whose values will be presented in the data component or whose value can be gathered from end users in the knowledge acquisition form.

The Presentation Designer

The Presentation Designer supports the specification of presentation styles for site view elements. As shown in figure 7.6, it relies on the site structure designer pane to choose the working page node, the page compositional structure pane to select the working site view element. The major component is the right pane, which allows the definition of templates and the association of templates to site view elements. It comprises *a template list pane* which allows the selection of a template from available one, *a template definition pane* which allows the browsing and specification of templates, and *a template saving pane* which allows the saving of templates.

As different types of templates have different features, the template specification pane varies accordingly. The right pane of figure 7.6 shows the specification pane for generic presentation styles. Figure 7.7 shows another two template specification panes, which are dedicated for the specification of presentation styles for site view elements with particular presentation requirements. Specifically, part (a) shows the specification pane for output elements which can be rendered using widgets. Part (b) shows the specification pane for data components which present dynamic data content retrieved from the underlying database for the specified class.

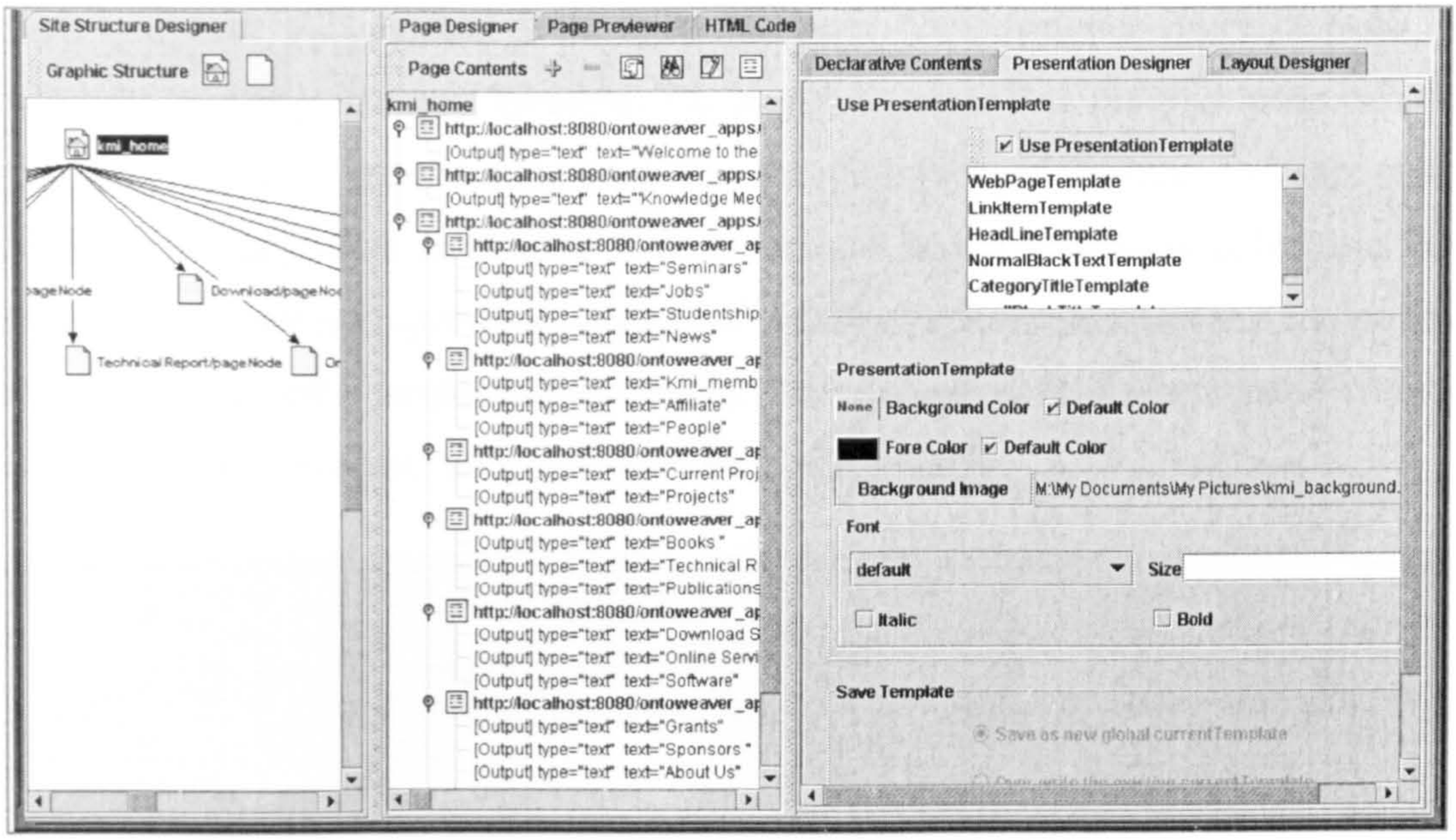
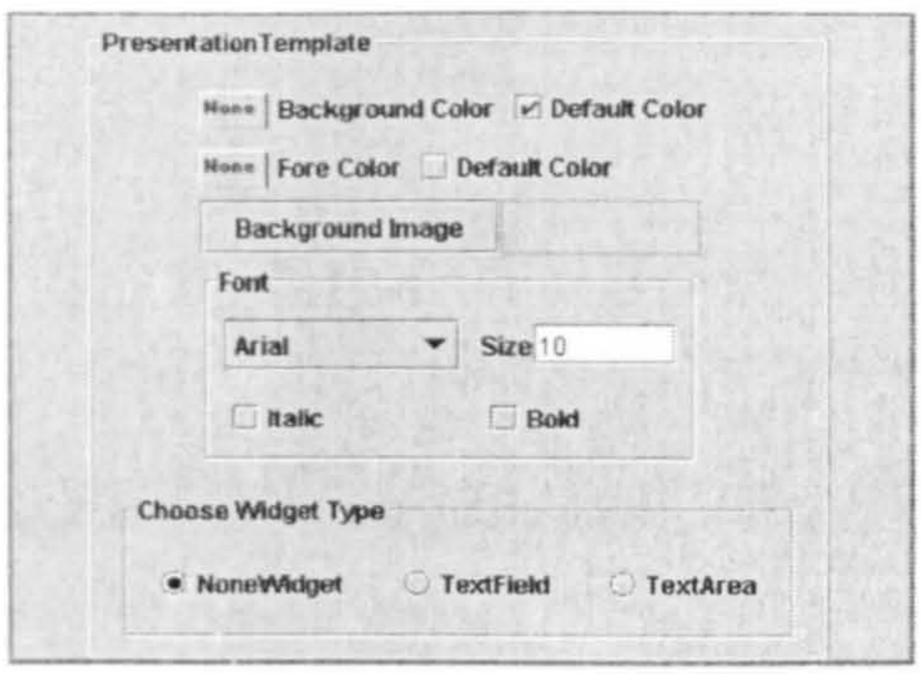
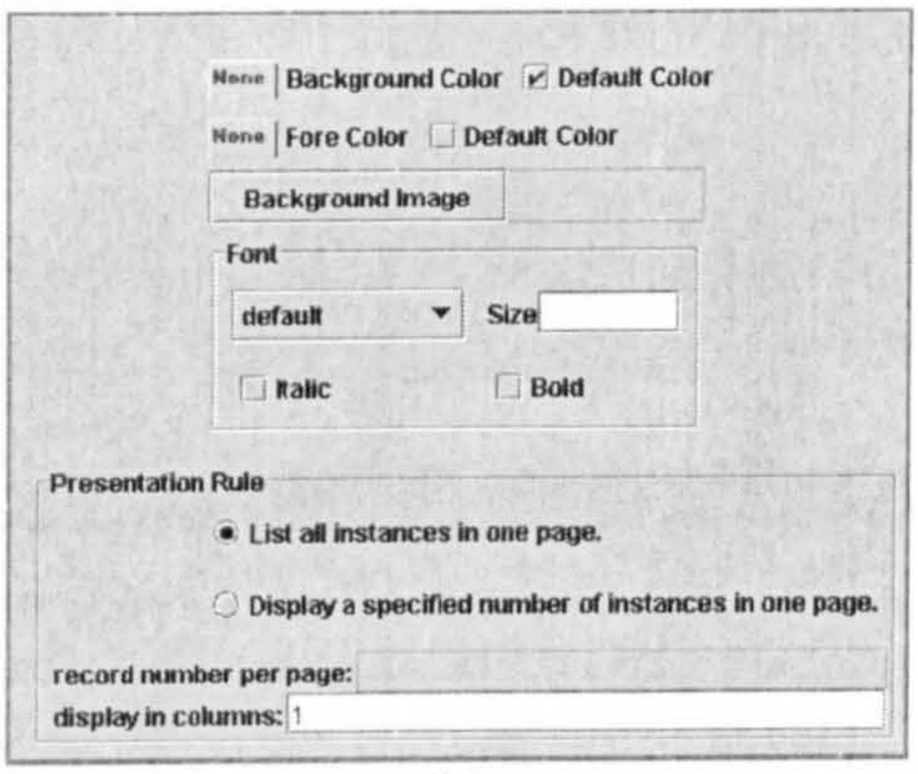


Figure 7.6 A screenshot of the Presentation Designer. It relies on the site structure designer pane to choose the working page node, the page compositional structure pane to select the working site view element. The major component is the right pane, which supports the specification of presentation styles. In this screenshot, the right pane shows the template specification pane for generic presentation styles.



(a)



(b)

Figure 7.7 The screenshots of the template specification panes for output elements and data components. Part (a) shows the specification pane for output elements. It allows the specification of a widget type for rendering an output element. Part (b) shows the specification pane for data components, which allows the specification of how to present dynamic data content.

The Layout Designer

The Layout Designer facilitates the specification of layouts for site view elements contained in web pages. Figure 7.8 shows a screenshot of the Layout Designer. Like the Presentation Designer, the Layout Designer relies on the site structure designer

pane to choose the working page node, the page compositional structure pane to select the working site view element. The major component is the right pane, which provides a layout specification pane for the definition of layouts for site view elements. As described in chapter 5, there are two layout constructs defined in the presentation ontology, specifying layout for atomic site view elements and composite site view elements respectively. Hence, the layout specification pane varies according to the features of the specified site view element.

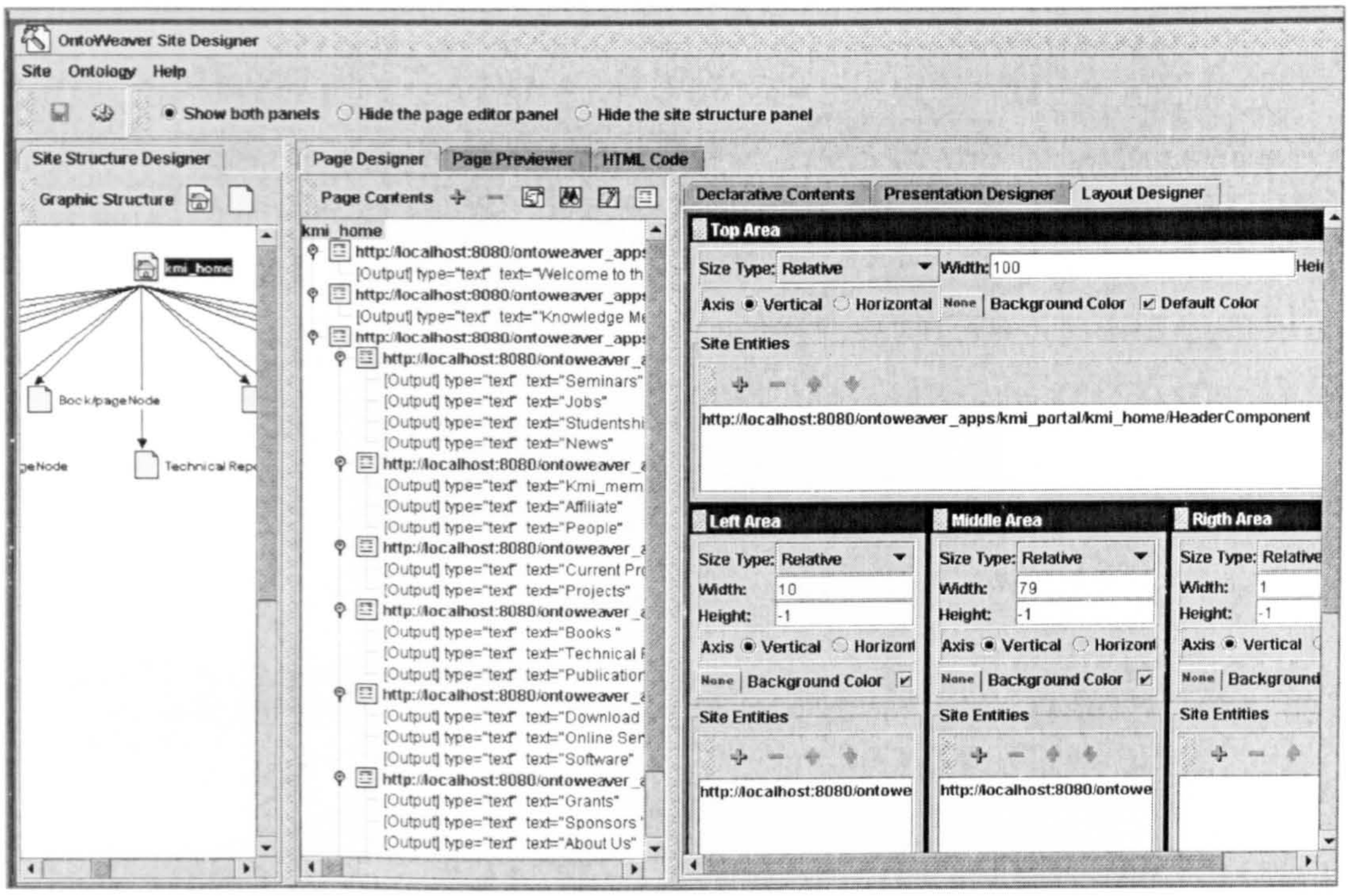


Figure 7.8 A screenshot of the Layout Designer. It relies on the site structure designer pane to choose the working page node, the page compositional structure pane to select the working site view element. The major component is the right pane, which allows the specification of layouts for the specified site view element. In this screenshot, the right pane shows the layout specification pane for a composite site view element, which allows web developers to position the sub elements contained in the working site view element into five sub-areas (i.e., top, left, middle, right, and bottom) and supports the specification of layout for each sub-area, e.g., width, height, and layout direction. In the case of specifying height or width as -1, which means no specific value is needed and a default value will apply in rendering.

The layout specification pane shown in figure 7.8 is for composite site view elements. According to the definition of the construct *ComponentLayout* specified in the presentation ontology, this layout specification pane provides five sub-panes which correspond to the five sub-areas of the construct and allow developers to position the sub-components contained in the working composite site view element

into appropriate sub-areas. For each sub area, the layout specification pane supports the further specification of layout, e.g., area size in terms of width and height and the direction of arranging the contained elements in terms of horizontal direction or vertical direction. In the case of using the number *-1* as width or height, no specific value is specified; the system relies on the web browser to render the size. Regarding atomic site view elements, the presentation ontology relies on the construct *TextLayout* to specify its layout in terms of alignment. Figure 7.9 shows the layout specification pane for atomic site view elements.

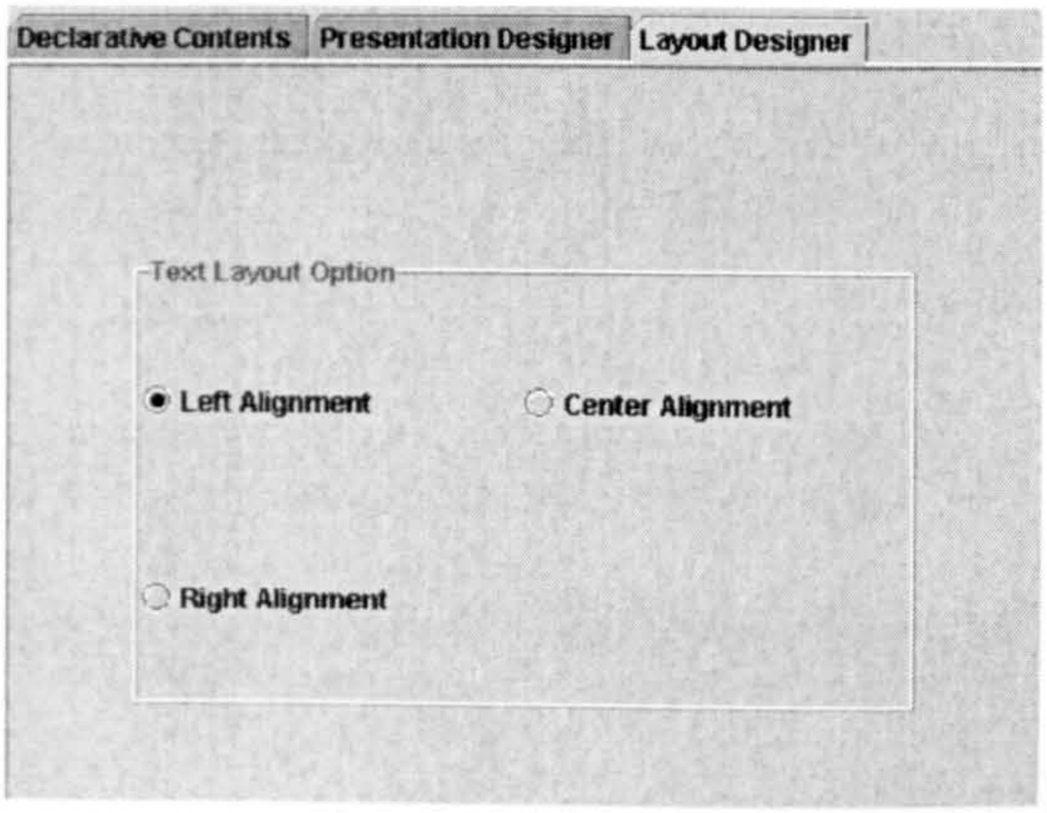


Figure 7.9 A screenshot of the layout specification pane for atomic site view element which supports the specification of how to arrange the specified site view element in terms of alignment.

7.1.3 The Site Builder

The tool *Site Builder* compiles the declarative specification of a web site into an implementation which can be accessed by end users. As described in chapter 4, the Site Builder employs a set of JSP code templates to render web site specifications. Figure 7.10 shows the compiling process which comprises three major steps: i) associating presentation specifications to site view elements, ii) applying design critiquing rules to validate both site view and presentation specifications, and iii) generating JSP code for each page node contained in the site structure according to site specifications and the pre-defined JSP code templates. Please refer to chapter 5 for the detailed description about the mechanism of generating implementation code from site ontologies.

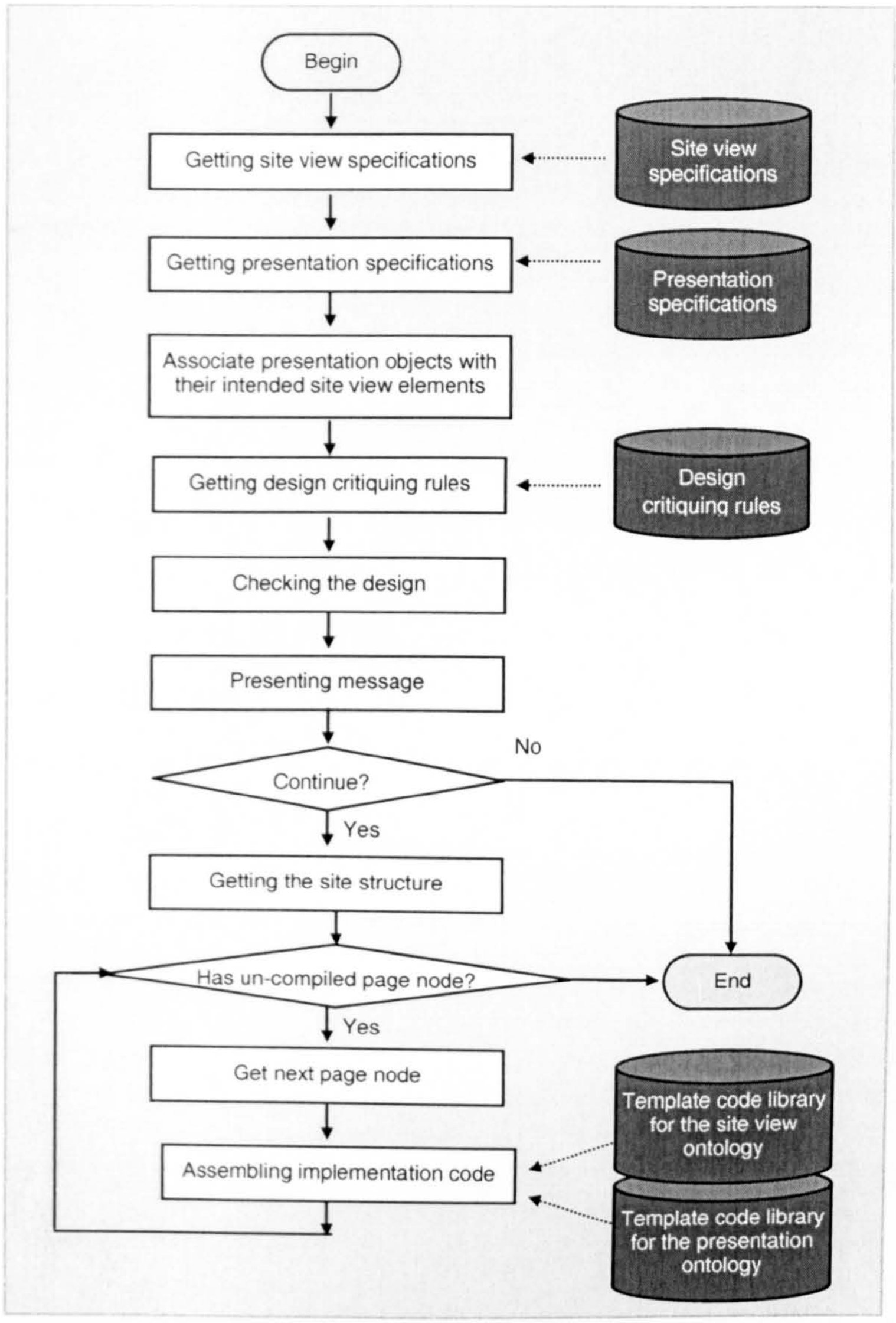


Figure 7.10 The process of compiling site specifications into implementations in the tool *Site Builder*

Now we look at the web site design critiquing facility. This is possible only when the design of the target web site is represented declaratively and can be reasoned upon. Critiquing rules thus can be applied to reason upon the site model to produce correction suggestions and recommendations for web site designers to improve their design. At the moment, a set of simple critiquing rules are embedded within the Site Builder, which check broken links, isolated web pages, and un-appropriate presentation instructions (e.g., illegible fonts). In future, further facilities will be

explored to allow web developers to define their own critiquing rules, thus offering customized critiquing services for web developers.

7.1.4 The Site Mapper

The tool *Site Mapper* provides two facilities. One facility is the *semi-automatic site re-engineering*, which maintains the conceptual relations between the site view components and the underlying domain ontology. As will be illustrated in section 7.3, the Site Mapper modifies the site view specifications according to the changes which have taken place in the underlying domain ontology, without losing the information specified by web site developers during the design process. Specifically, the tool adds new components associated with newly added domain entities, removes components associated with the domain entities which have been removed in the domain ontology, or doing both when domain entities have been modified. Other components remain unchanged.

The second facility that the tool *Site Mapper* provides is the automatic site specification generation, which produces a default site specification by instantiating the site view ontology using the domain ontology. As will be described in section 7.4, the automatic generation process starts from scanning the hierarchy structure of the domain ontology and produces a default site structure for the target web site. It then maps each class to a set of page nodes which include a page for data presentation, a page for data acquisition, and a page for data querying.

7.1.5 The Site Customizer

The OntoWeaver Site Customizer allows web developers and web administrators to carry out customization design. As discussed in chapter 6, the Site Customizer comprises two major components. One is the user group customization pane, which supports the creation and management of user groups and user group specific site models. The other is the rule-based customization pane, which facilitates the specification of customization rules. These components will be illustrated later in section 7.2.

7.1.6 The OntoWeaver server and server-side components

As mentioned earlier, the server-side components of the OntoWeaver tool suite provide services for accessing and manipulating ontologies stored in the back-end knowledge warehouse. These services are implemented as web services, which can be invoked locally as well as remotely. Front-end tools rely on these services to access and manipulate the specified ontology. For example, the Ontology Editor relies on the services provided by *ontology readers* and *writers* to access and manipulate the back-end domain ontologies. Likewise, the Site Designer relies on *a set of specification readers and writers* to access and manipulate the site view specifications.

The services have a unique accessing interface, which is using *a service name* to identify the name of the service component in question and *a parameter object* to describe the parameters needed for the service. They are invoked by means of the OntoWeaver server with the specification of the intended service name and a valid parameter object. Figure 7.11 illustrates a service example, which gets the site view model from the specified URL of a site view specification document represented in RDF. The result of this service is a Java object, which represents the model of the site view specifications. The OntoWeaver applications can get a site view model from a remote RDF document by calling this service.

serviceName: getSiteViewModelFromRDF

parameter: the URL of a site view specification document (in RDF)

result: a site view model

Figure 7.11 The service interface for getting a site view model from the URL of a site view specification document represented in RDF.

The architecture makes the infrastructure of the tool suite flexible. The server-side services are re-usable, and thus provide support for more than one application. Furthermore, it makes possible for the OntoWeaver visual tools (e.g., the Ontology Editor, the Site Designer, and the Site Customizer) to become available online in the future, as it separates visualization from the management of the underlying data sources.

7.1.7 The Online Page Builder

The Online Page Builder generates a customized web page dynamically on the fly when a user makes a web page request. It relies on the server-side components to retrieve the user group specified site model which is designed particularly for the user group that the user falls in. It then invokes the customization engine to generate a customized site model. Thereafter, the Online Page Builder calls the Site Builder to generate a customized web page on the fly by compiling the customized specification of the page node in question into web page implementations.

7.2 An example: building the KMi web portal

The KMi web portal is a data-intensive web site example, which has been illustrated in chapter 5. The essential requirements of the web portal comprise: i) presenting information about the Knowledge Media Institute, such as *research projects*, *events*, *publications*, and *people*, and ii) presenting customized views to end users according to their user groups and profiles. A typical process of building a data-intensive web site in OntoWeaver involves the following steps:

- *Requirements collection and analysis* identify the objectives of the target web sites, data characteristics, user categories, user requirements, and so on.
- *Domain ontology design* develops a data model to abstract the problem domain.
- *Site structure design* defines the site structure for the target web site.
- *Page content design* produces detailed content for each page node defined in the site structure.
- *Presentation design* specifies presentation styles and layouts for site view elements in the site view model.
- *Customization design* identifies and specifies customization requirements in terms of user ontology and customization rules.

In this section, we will illustrate how the OntoWeaver tool suite can be used to support these design activities.

7.2.1 The domain ontology design

As mentioned in the beginning of this chapter, domain experts are responsible for the design of domain ontologies. The domain expert of the KMi web portal first analyses the information requirements of the problem domain and draws a hierarchy structure of the main concepts of the domain ontology which has been shown in chapter 5. He then uses the Ontology Editor to create the domain ontology. As mentioned earlier, there are two ways of creating ontologies in the Ontology Editor: i) creating ontologies from scratch, and ii) importing an existing ontology and adapting it to meet particular requirements involved with the problem domain. In the context of the KMi web portal, the domain expert chooses the latter option. Figure 7.12 shows the user interface that allows the import of pre-existing ontologies (in RDF Schema).

Creating class entities and property entities

Now suppose the domain expert is creating a class entity called *Software*, which abstracts the software products that KMi produces. Three existing properties need to be associated with the class. They are *title*, *description*, and *web_address*. As the class *Software* has no super class in the domain ontology, the domain expert clicks the root of the *class tree* and presses the button *"New Child"* on the right pane. A class specification form is presented. The domain expert defines the class Software and associates the appropriate properties with it by means of the property list pane contained in the form. Figure 7.13 (a) shows the screenshot of the class specification form. Figure 7.13 (b) shows the dialog that allows the association of the selected properties with the class entity in question. The process of creating property entities is similar to that of the creation of class entities in the Ontology Editor.

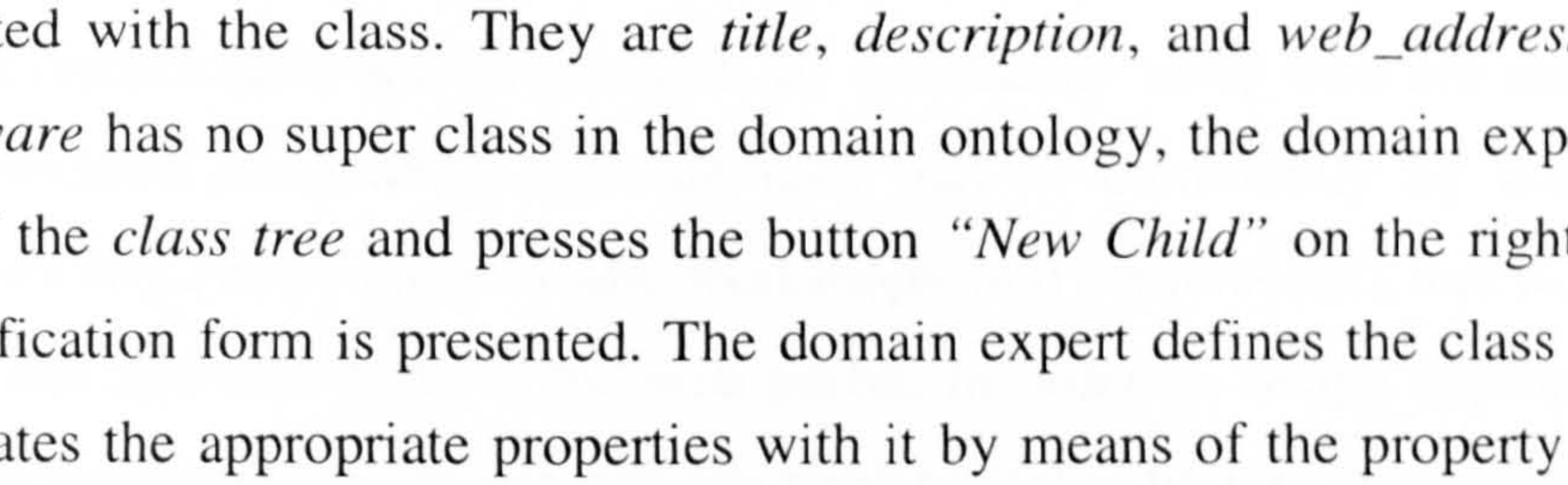


Figure 7.12 The user interface that allows the importation of an online ontology

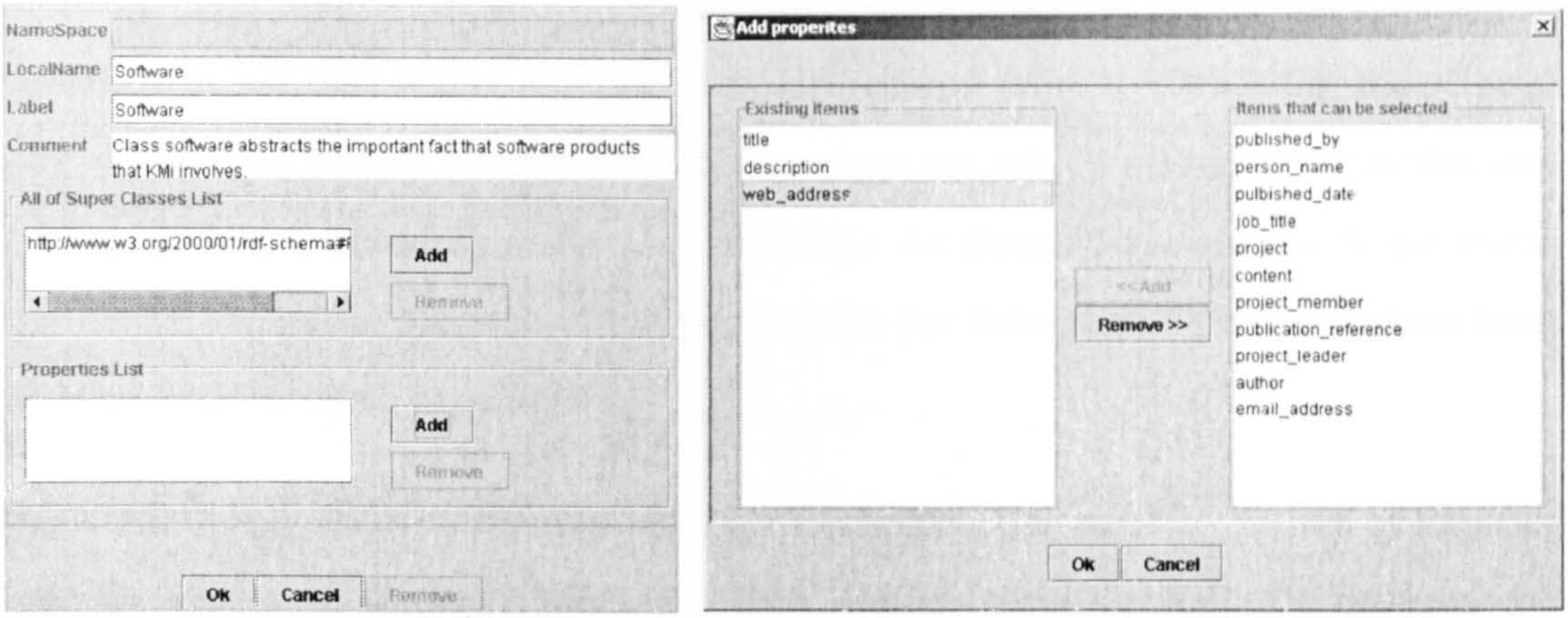


Figure 7.13 The user interfaces for creating a new class entity or editing an existing one. Part (a) shows the class specification form. Apart from the specification of the associated class entity, this form also allows domain experts managing super classes and the associated properties. Part (b) presents the dialog that allows the management of the associated properties for the class entity in question.

7.2.2 Designing a general view for the KMi web portal

In the context of the KMi web portal, there are three categories of potential audiences, including i) *general users* who visit the web portal to get the knowledge of the KMi organization and its research, ii) *community users* who are members of the KMi research community and can take part in community by contributing relevant news (e.g., news on seminars, workshops, and conferences), and iii) *internal users* who can add new facts to the web portal. In order to design appropriate site views for different user categories, the site designer takes two major steps to achieve the task: i) designing a general view of the KMi web portal and ii) specifying customization requirements. In this section, we explain the first step. The second step will be explained in section 7.2.4.

Creating the site structure

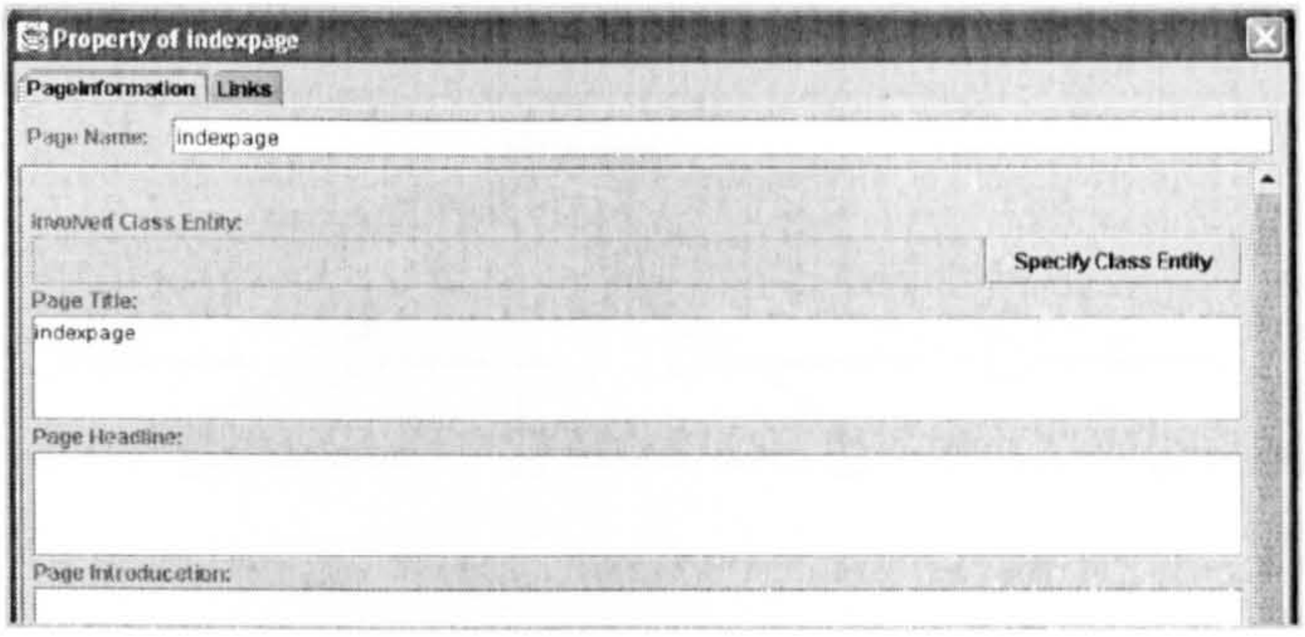
Creating site structures can be achieved by using the *Site Structure Designer*. During the site structure design process, the site designer of the KMi web portal first specifies the KMi domain ontology as the underlying data model using the provided menu item *"Specifying the domain ontology"* contained in the menu group *file* of the Site Designer. He then uses the site structure designer pane (as shown in figure 7.3)

to create page nodes and links. Specifically, he presses the index page button contained in the tool bar of the site structure design pane to create the index page node of the web portal. A page node dialog pops up, which allows the specification of the page node, including *meta-data* and *links*. As discussed in chapter 5, the meta-data define the initial purpose of the page node; the links define the navigation from the page node to other page nodes.

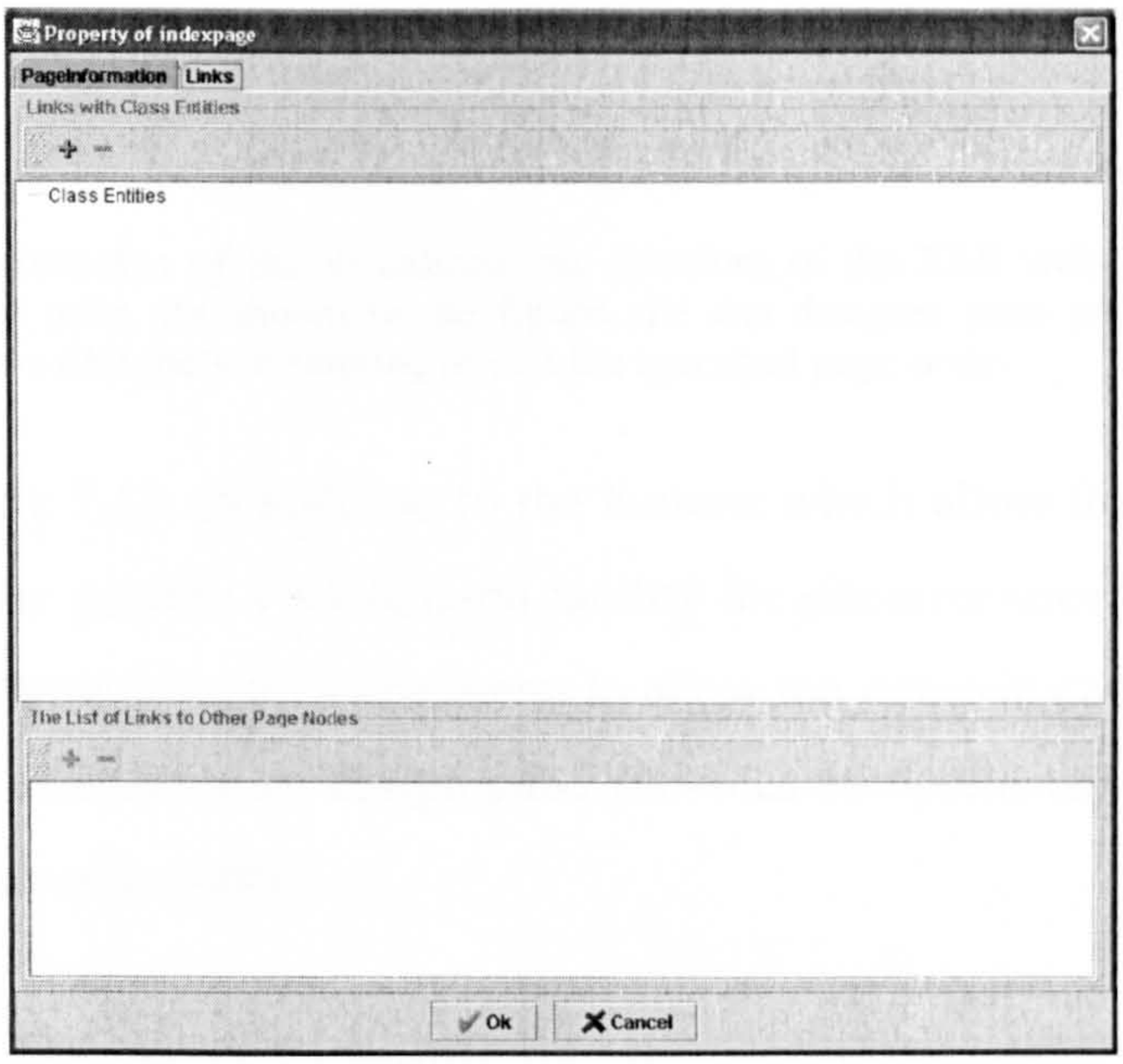
The site designer defines the meta-data for the index page node by means of the user interface shown in figure 7.14(a). He then defines the link structure between the index page node and all other page nodes which intends to present instances of specific domain class entities. This is achieved by specifying conceptual links with domain class entities. Figure 7.14 (b) and figure 7.14 (c) show the user interfaces for the specification. By specifying a conceptual link, a web page which intends to present instances of the specified class is created. At the same time, a link is created which allows navigation from the page node in question, which is the index page node in this context, to this newly created page node.

When all the conceptual links have been defined for the index page node, the site designer gets a graphic visualization of the initial site structure in the site structure designer pane. All page nodes that intend to present instances of specific domain class entities are created after the specification of conceptual links. But other page nodes that present static information have not been created yet, e.g., the page node which presents the major contacts of the KMi research institute. The static page nodes can be created by pressing the button . The page node specification procedure remains the same as the process of the index page node discussed above. The link relations between the index page node and the newly created page nodes can be added by editing the index page node. Figure 7.15 shows a screenshot of the site structure of the KMi web portal.

(a)



(b)



(c)

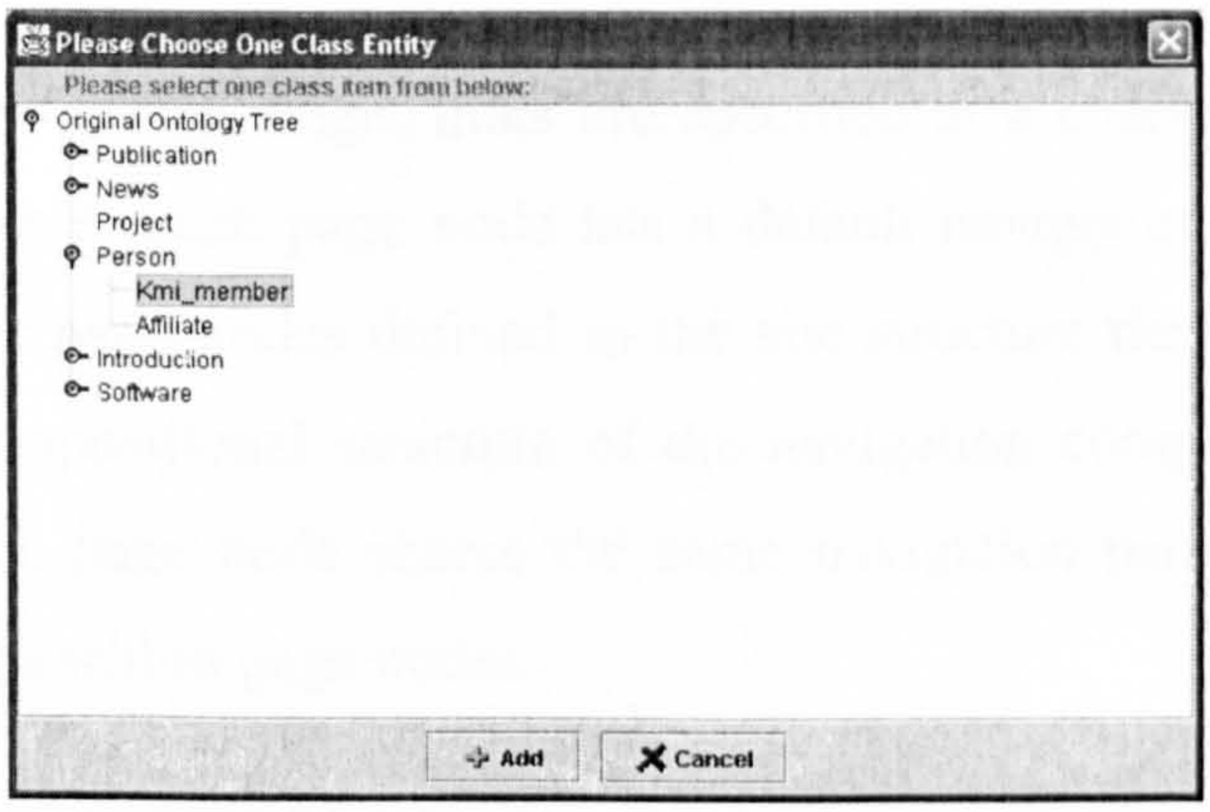


Figure 7.14 Screenshots of the user interfaces which support the specification of the associated page node, including specifying meta-data and links. Part (a) shows the pane that allows the specification of meta-data. Part (b) shows the pane that allows the definition of links between the current page node and page nodes which either already exist or be generated from the specified class entity. Part (c) shows a dialog that allows the selection of a class entity. Along with the selection, a new page node is created, which intends to present instances of this class entity and a link is created which allows navigation from the page node in question to this newly created page node.

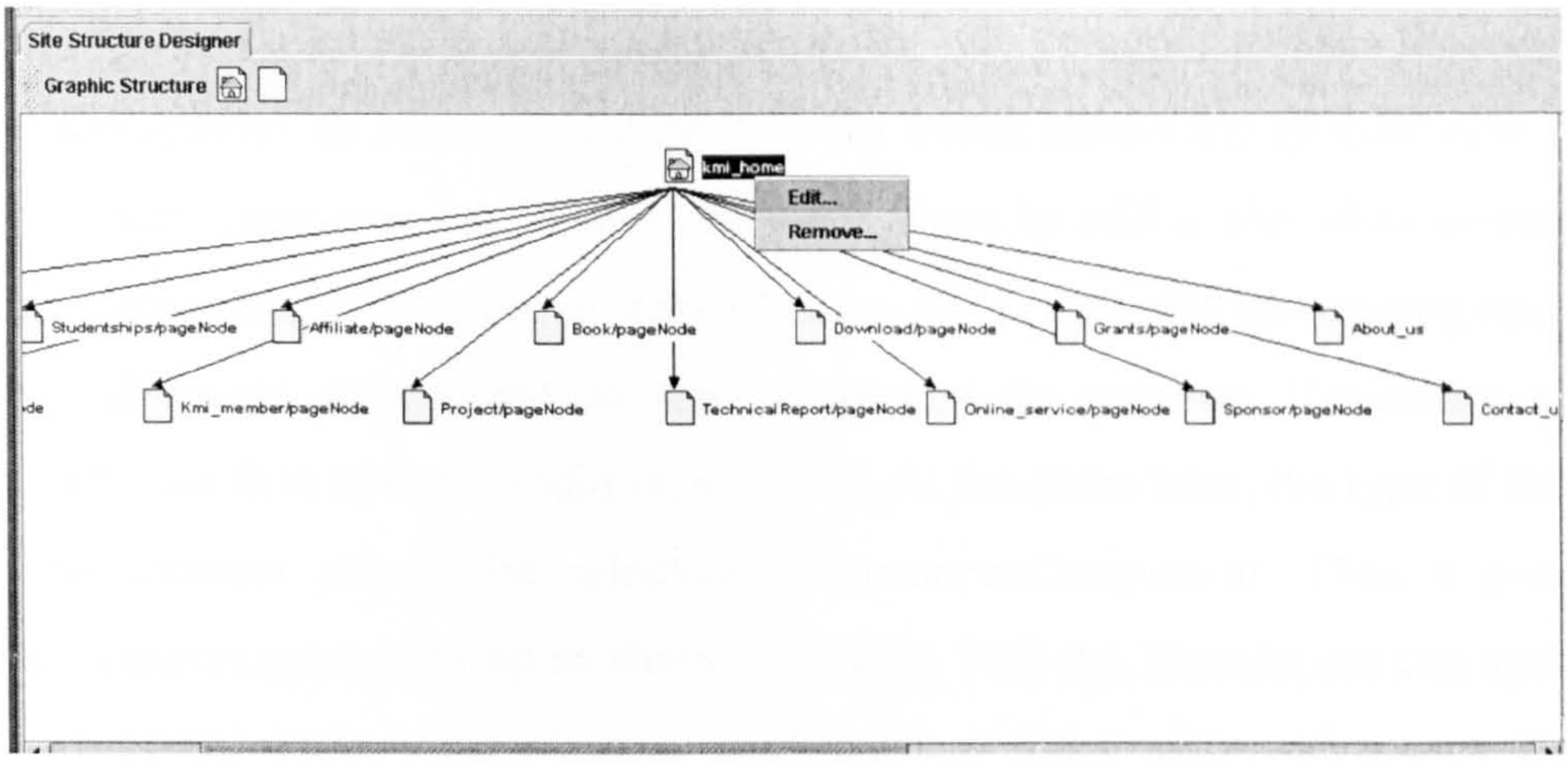


Figure 7.15 A screenshot of the visualized site structure of the KMi web portal in the site structure designer pane. As shown in the figure, the site designer pane provides right-click menu which allows designers to remove or edit the specified page node.

As shown in figure 7.15, in addition to the buttons which allow the creation of new page nodes and the graphic visualization facility for site structures, the site structure design pane also provides right-click menu to allow the removing and editing of page nodes. Thus, the site structure design pane provides comprehensive support for the specification of site structures.

Organizing links into different categories

At the site structure design stage, links are specified at a coarse grained level. As described in chapter 5, each page node has a default navigation component which holds links to other page nodes defined in the site structure design process. Figure 7.16 shows the compositional structure of the navigation component of the index page node. As each page node shares the same navigation pattern, the navigation component is shared within page nodes.

Now the site designer works on organizing them into six categories: *News*, *People*, *Projects*, *Publications*, *Software*, and *About KMi*. The strategy is to create one sub-component for each category within the navigation component (which is the original component that contains the links) and then to move the links from the navigation component to the corresponding components. Figure 7.17 illustrates the buttons which allow the manipulation (adding/removing) of compositional structures of site

view elements. To create a new component, the site designer presses the button ✚ and a form appears as shown in figure 7.18 (a) which allows the specification of the type of a site view element. There are three ways to add a site view element: i) creating a new one, ii) re-using an existing one, and iii) moving an existing one from its original parent component to the component in question. To create a new component, the first option should be selected. At the same time, the type of the new site view element should be selected as *ResourceComponent*. Thus a pane for defining a component pops up as shown in figure 7.18 (b). Developers can specify a title, conceptual links with classes, and physical links with web pages for the component. Using these facilities, the site designer creates one sub component for each link category within the navigation component.

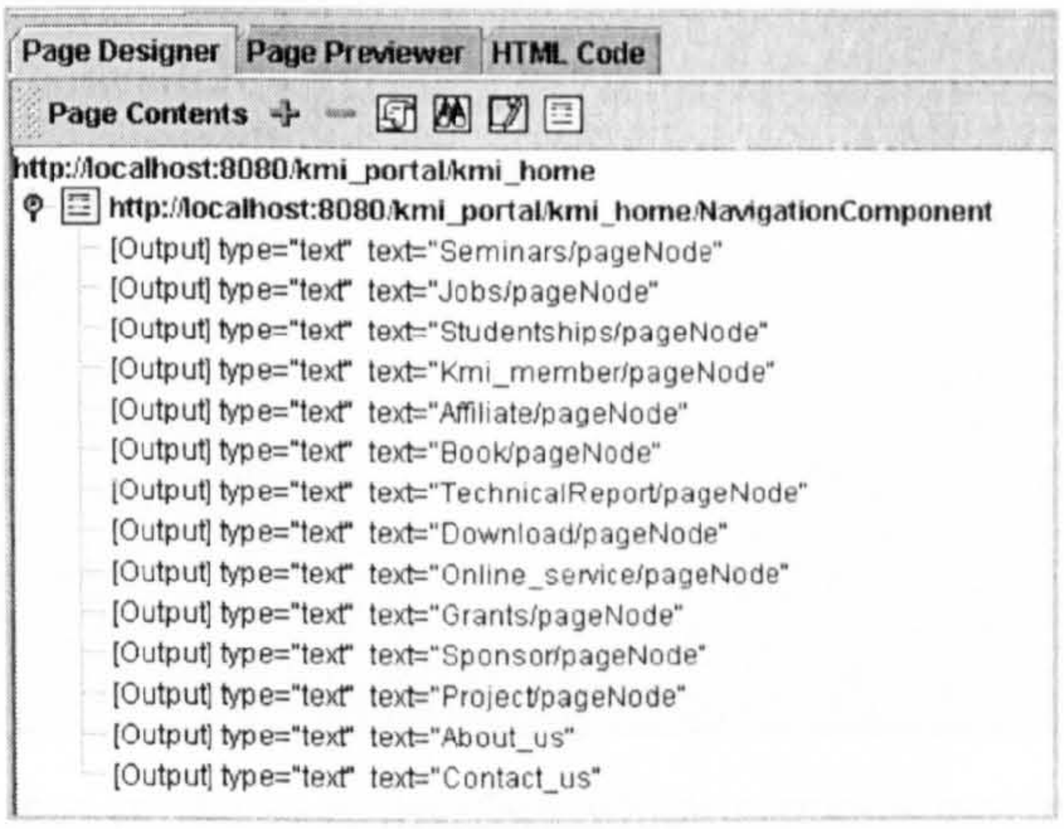


Figure 7.16 The compositional structure of the navigation component created in the site structure design phase

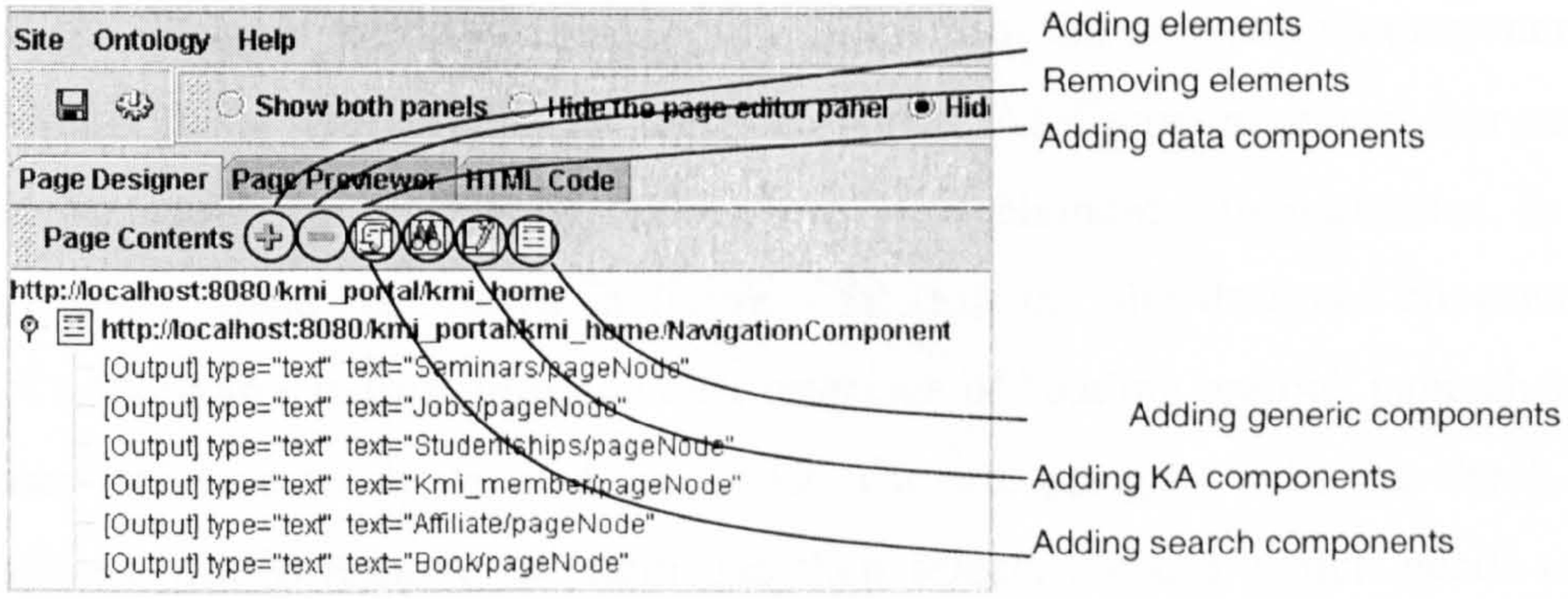
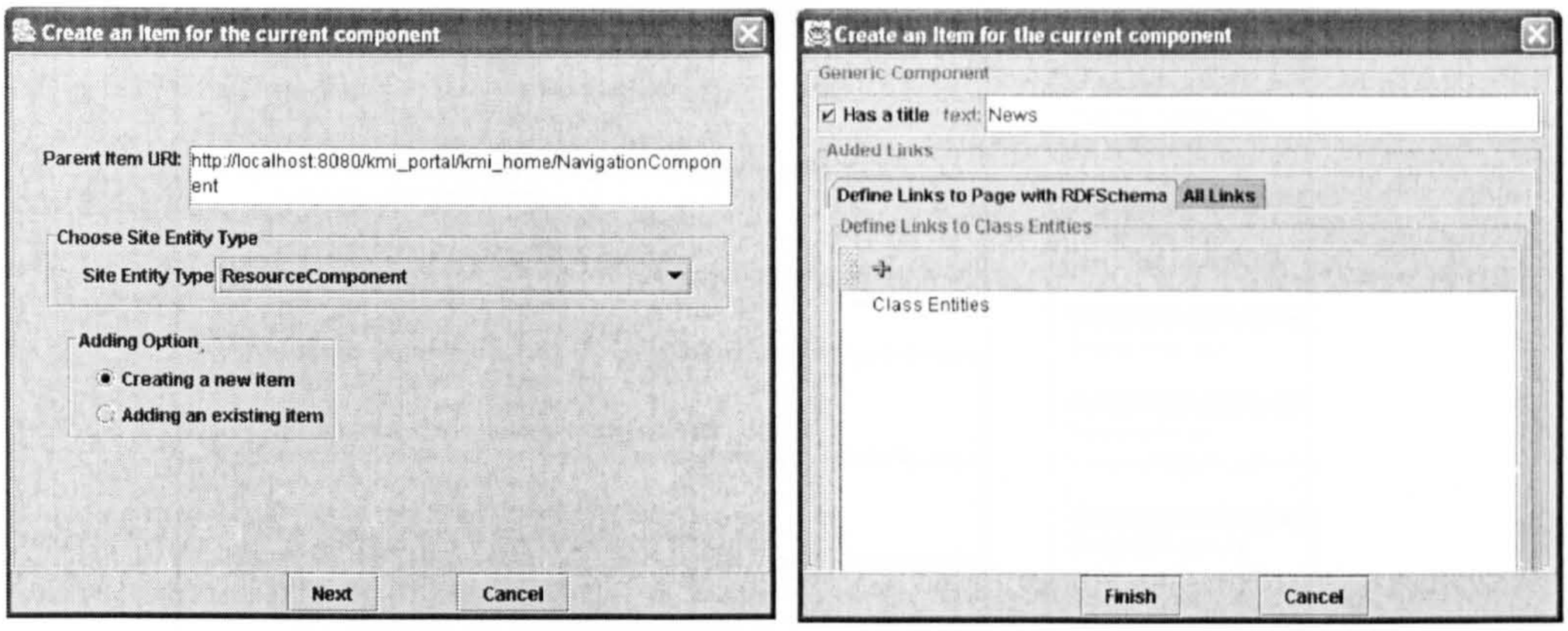


Figure 7.17 An illustration of buttons contained in the page compositional structure pane. They allow the manipulation of the compositional structures of site view elements.



(a) (b)

Figure 7.18 Screenshots of the user interfaces which allow the adding and creation of new components. Part (a) shows the option that should be selected for creating a new resource component. Part (c) shows the form for defining information for components.

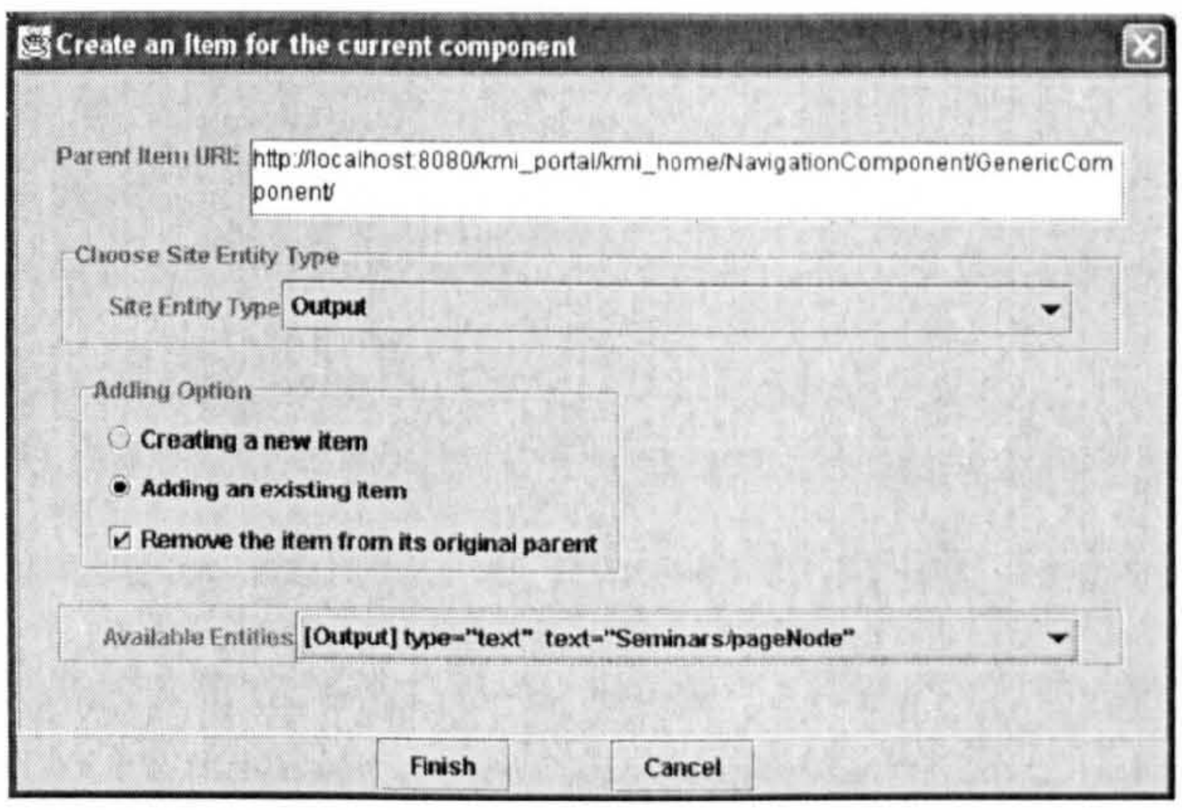


Figure 7.19 A screenshot of the user interface which supports moving the specified output element to the component in question. The moving entity type is specified as *Output*; the adding option is moving an item to the current component; and the selected site entity is the output element which is labeled as "Seminars/pageNode".

Now the site designer works on moving the links from the navigation component to the sub-components which represent their categories. As discussed above, this can be achieved by using the facility of adding site view elements. In particular, in the "adding option" pane (as shown in figure 7.18 (a)), the site designer chooses the option of adding an existing item, the user interface of "adding option" pane changes to the user interface as shown in figure 7.19. The site designer ticks the check box which allows him to remove the item. He then selects the link which needs to be moved. Thus, the site designer is able to associated links to their corresponding categories. Figure 7.20 (a) shows the resulting organization of the navigation component. Figure 7.20 (b) shows the preview of the web pages.

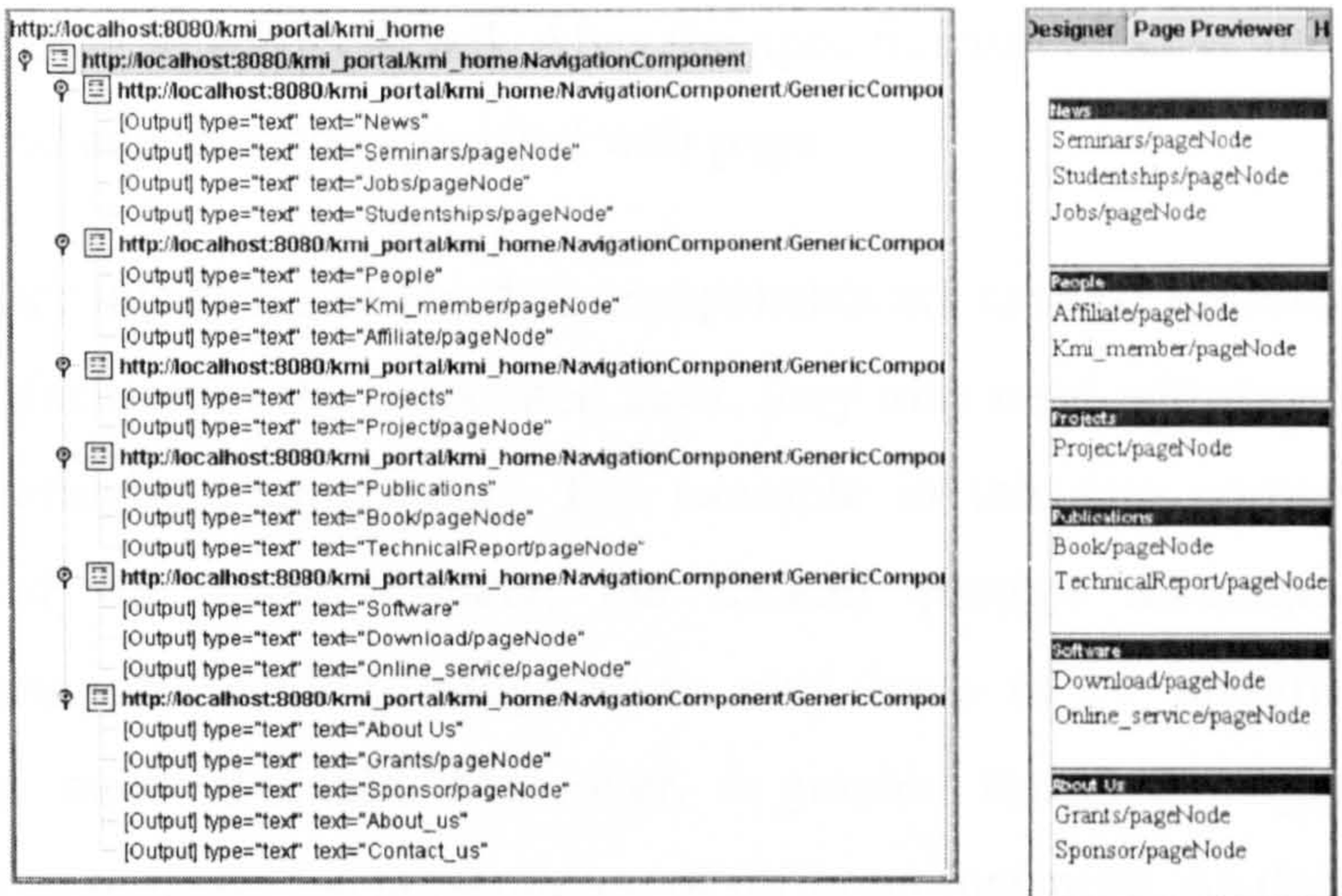


Figure 7.20 The resulting structure of the navigation component. Part (a) shows the compositional structure of the navigation component after which has been re-organized. Part (b) shows the preview of the navigation component. Each link category has a title and contains a number of links.

Modifying links

As shown in figure 6.20(b), the texts of links are created automatically according to the specification of the link structure at the site structure design stage. They can be modified by means of the declarative content pane of output elements, which visualizes links. This pane appears automatically as shown in figure 6.4 (a) when developers choose an output element in the compositional structure pane.

Adding dynamic content to web pages

Dynamic content is required in the KMi web portal in order to present instances of domain class entities in web pages. Adding such dynamic content can be achieved by means of adding data components. Suppose the site designer is working on adding dynamic content to the project web page. He selects the web page from the site structure graph, and presses the button on the tool bar of the page compositional structure pane which has been illustrated in figure 7.17. A dialog then pops up, which contains a declarative pane (as shown in figure 7.5 (a)) to allow the definition of the new data component. In particular, the site designer can specify a list of slots whose

values are going to be displayed. After the specification process, the data component is created and added to the specified web page.

After the user interfaces of the data components are created automatically according to the specification of the associated slots, they may need adjustments or extensions to meet particular requirements. For example in the data component presenting instances of the class *Project*, the default prompt messages for the slots *project_name*, *description*, and *picture* may have to be modified to make the information more readable. Moreover, a graphic image is required in the data component to separate the presentation of different instances. As described in chapter 5, these adjustments can be easily achieved as all components are specified declaratively and are thus available for modification. The OntoWeaver Site Design supports this task. For example, unnecessary output elements can be removed by using the button − for removing site view elements. New output elements can be added by using the facility for adding site view elements discussed above. Figure 7.21 shows the screenshot of the project web page after the adjustments.

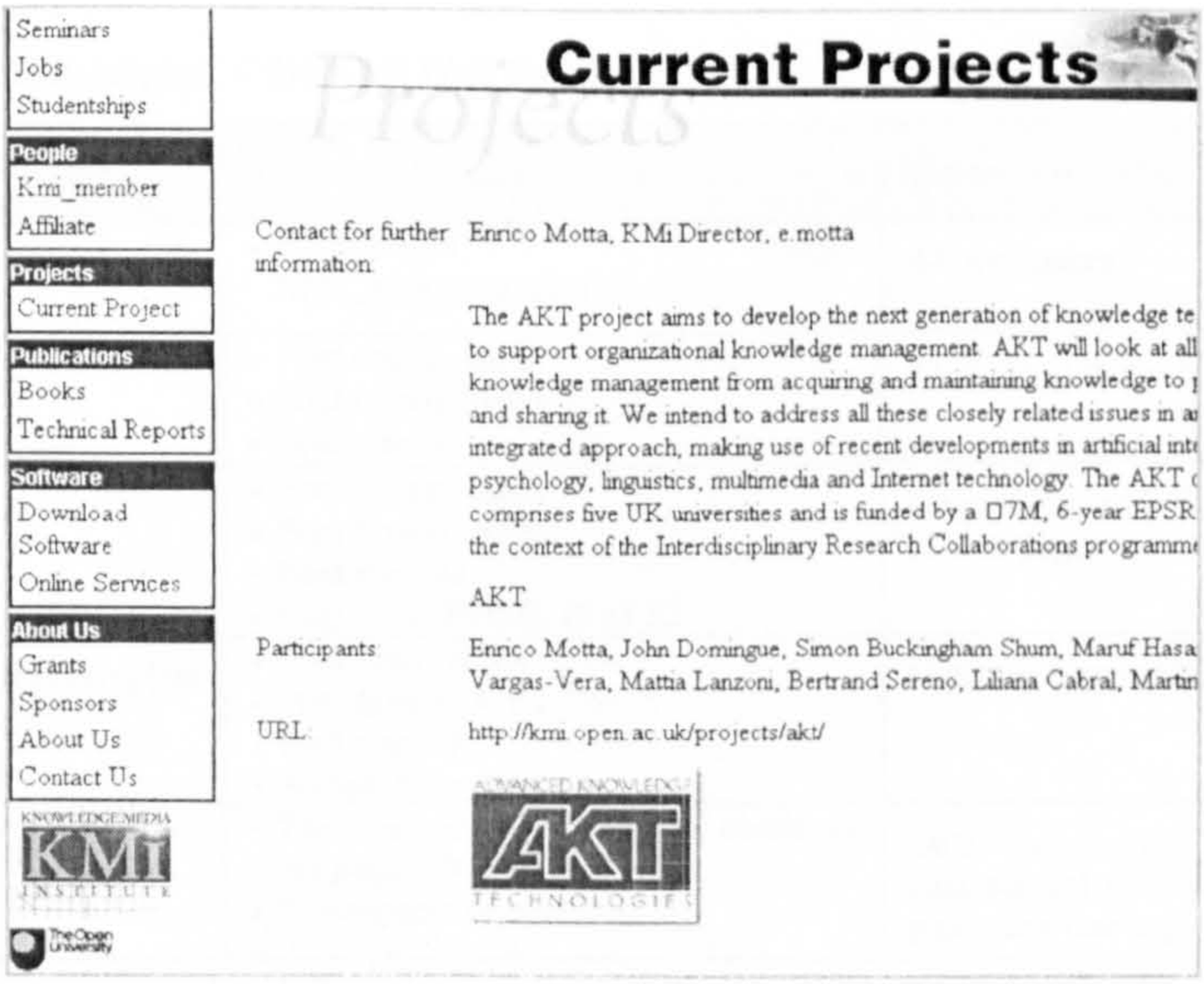


Figure 7.21 A screenshot of the project web page

7.2.3 The layout design

The layout designer takes three steps to design presentation styles and layouts: i) creating a set of templates which describe the presentation styles for web pages and their components, ii) attaching templates to site view elements, and iii) specifying organizations for site view elements.

Specifying presentation styles

As described in chapter 5, OntoWeaver distinguishes three kinds of presentation templates: *PresentationTemplate,* which abstracts visual appearances of web pages, resource components, and output elements, *WidgetPresentationTemplate,* which models visual appearances for user interface elements that are involved with widgets, and *DataComponentPresentationTemplate,* which describes presentation styles for components that publish dynamic data content. In the context of the KMi web portal, the layout designer defines a set of presentation templates for site view elements. Table 7.1 lists some examples:

Table 7.1 The template examples of the KMi Web Portal

Presentation Template Name	**Template Feature**	**Target Site View Elements**
WebPageTemplate	• Background image: KMi_background.gif	All web pages
LinkItemTemplate	• Font colour: black • Font family: Arial • Font size: 9	All links in the web portal
HeadLineTemplate	• Font colour: black • Font family: Arial • Font size: 24 • Font style: BOLD, ITALIC	Output elements for presenting headlines of web pages
NormalDynamicBlackTextTemplate	• Font color: black • Font family: Arial • Font size:10 • Widget type: none	Dynamic output elements
DataTemplate	• The number of data records displayed per page: 10 • The number of columns: 2	Data components that publish dynamic data content in more than one column and more than one page

As already shown earlier in figure 7.6, the OntoWeaver Presentation Designer allows the definition of templates and the attachment of templates with site view elements. The left pane allows the selection of the working site view element. The right pane facilitates the specification of presentation styles.

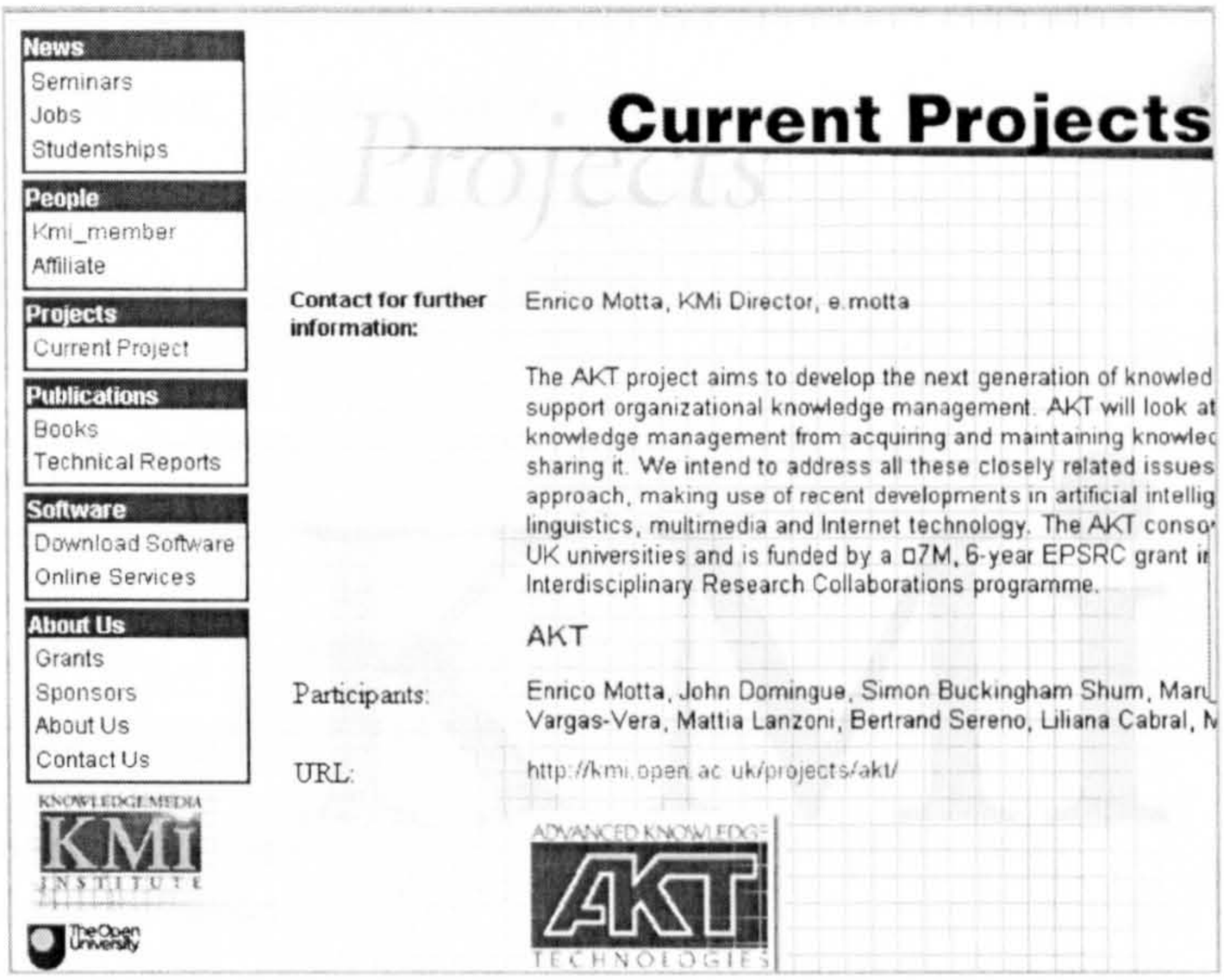


Figure 7.22 A screenshots of the project web page after the specification of presentation styles

Imagine the layout designer is now working on creating a presentation template for output elements. He selects one of the output elements that the template works on, defines the template in the *template definition pane*, and saves the specification. Thereafter, this template is ready for other output elements. Attaching a template with a site view element can be easily achieved by clicking on the site view element in the left pane of the Presentation Designer and selecting the template from the list of available ones contained in the right pane. In this way, the layout designer achieves the tasks of defining presentation styles for the web portal. Figure 7.22 shows the screenshot of the new version of the project web page.

Specifying layouts

Now the layout designer focuses on re-organizing the layout of web pages. This is supported by the tool *Layout Designer*. As described earlier in section 7.1.2, the left pane of the Layout Designer allows the selection of the working site view element. The right pane allows the specification of a layout for the working site view element. Imagine the layout designer is working on re-organizing the *project data component* which presents the instances of the class *Project*. As described in chapter 5, the OntoWeaver presentation ontology divides an area of a composite site view component into five sub-areas and thus can position its sub elements into different

areas according to requirements. The layout of each sub-area can be specified. Furthermore, the layout of each sub-element can be further specified in the same way. Hence, complex layouts can be specified for web pages. The tool Layout Designer supports this specification method.

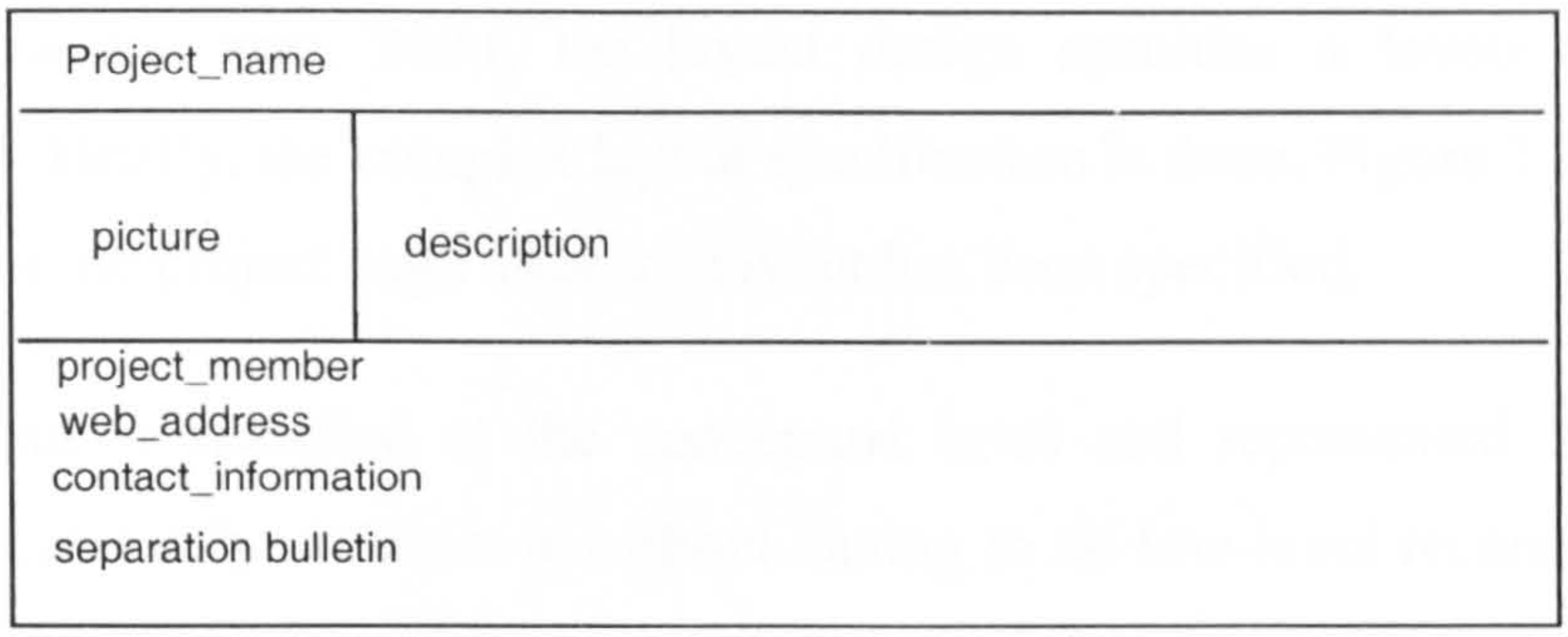


Figure 7.23 A layout of the project data component

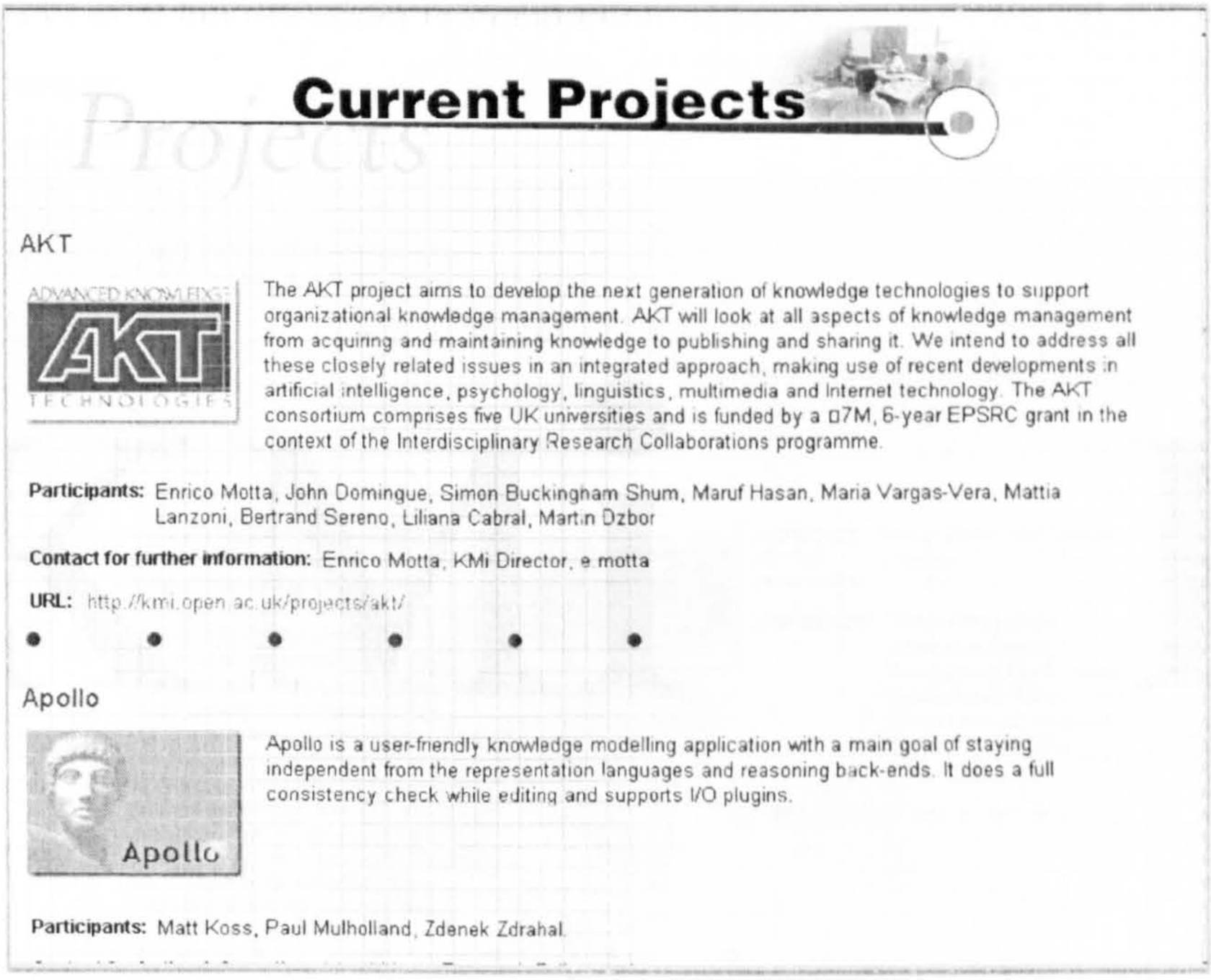


Figure 7.24 A screenshot of the project page after the layout design

As shown in figure 7.22, the default layout of this component is to present all sub-components in one area in a vertical sequence. The layout designer designs a new layout, which is illustrated in figure 7.23. To specify this layout, the layout designer moves the component that presents content of the slot *project_name* to the top area

by pressing the adding button of the top area and chooses the component. In this way he moves the component *picture* to the left area and the components *project_member*, *contact_information*, *web_address*, and the picture of separation bulletin to the bottom area. Thereafter, he adjusts the sequence of the site view elements positioned in the bottom area according to the requirements and specifies size for each sub area. Next, the layout design specifies a layout for its sub-components. Finally, the complex layout specification is done. Figure 7.24 shows the screenshot of the project page after the layout has been specified.

As the layout is specified at the conceptual level and represented declaratively, developers can easily maintain it without having to do low-level reconstruction. For example, we can easily re-arrange the layout of the project data component by placing sub-elements into different areas. Figure 7.25 illustrates how to position sub elements into five sub-areas of a component to form another layout for the project data component.

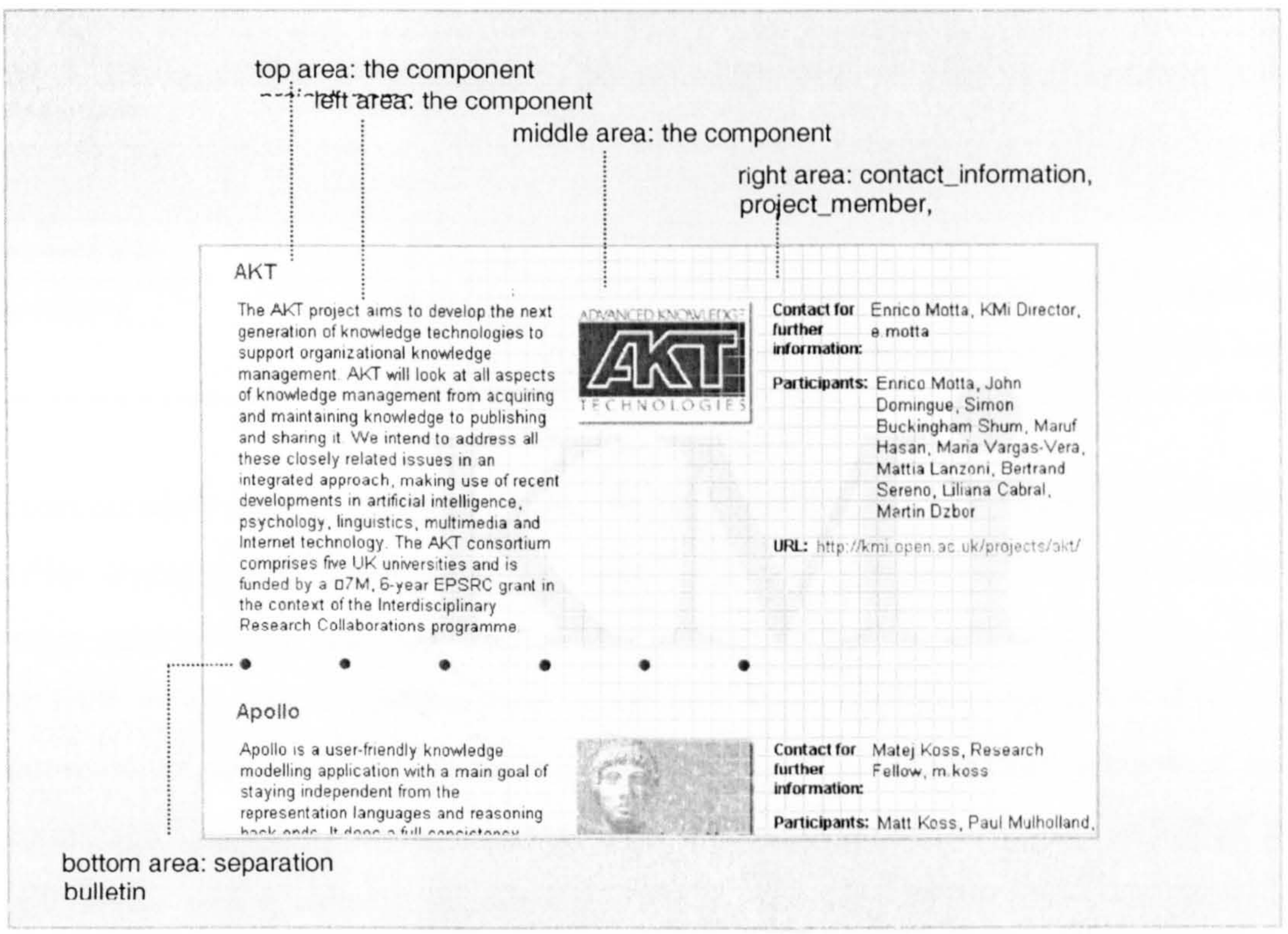


Figure 7.25 The illustration of how to position sub elements of the project data component into five sub-areas to form another layout.

7.2.4 User group specific customization design

As clarified in chapter 6, OntoWeaver supports user group specific customization design by allowing the creation of different site views for different user groups and user individual specific customization design by allowing the specification of customization rules which define when and how to perform customization. In this section, we focus on the user group specific customization design for the KMi web portal. The user individual specific customization design will be explained in the following section.

As mentioned earlier, there are three groups of potential users in the KMi web portal. They are *general users*, *community users*, and *internal users*. The general view of the web portal which targets the first user category has been specified through the steps discussed above. Views for the other user groups can be created upon the basis of the general site model. Table 7.2 shows the requirements for deriving particular views for the other two user groups.

Table 7.2 The requirements of deriving customized views of the KMi web portal for community users and internal users

User Group	**Customization**
Community users	• Adding a web page, which allows the submission of news • Adding a link to the news web pages
Internal users	• Adding a set of web pages, which allow the creation of new facts, e.g. Projects and Publications • Adding relevant links into the relevant web pages to allow the navigation to the newly created web pages.

Customization design is supported by the tool *Site Customizer* which as mentioned earlier comprises two major components: the user group specific customization design pane and the rule-based customization design pane. As shown in figure 7.26, the user group customization pane comprises three major components: i) *a user management pane* (located in the left side), which supports the management of user groups and individual users, ii) *a site model assignment pane* (located in top of the right side), which allows the specification of site models for the user group in question, and iii) *a site designer pane* (located in the bottom of the right side), which allows the browsing and editing of the specified site models. Now we illustrate how to create user group and user group specific site views.

Creating user groups and specifying site models

To create the user group *community users*, the site designer chooses the root of *the user group tree* shown in the left pane in figure 7.26 and clicks the button ✣. A form then pops up as shown in figure 7.27 (a), which allows the definition of the new user group. In particular, this form allows the specification of the user group specified site view model and presentation model. Figure 7.27 (b) shows the dialog that allows developers making use of the pre-defined site specifications to specify site models for the user group in question. The site designer uses the general site view model, which is stored in the file *kmi_portal.rdf*, as the starting point of the group specified site view model, and saves it in another file. In this way, the group specified site view model is based on the general view model and the modification on the model will not affect the original site view model.

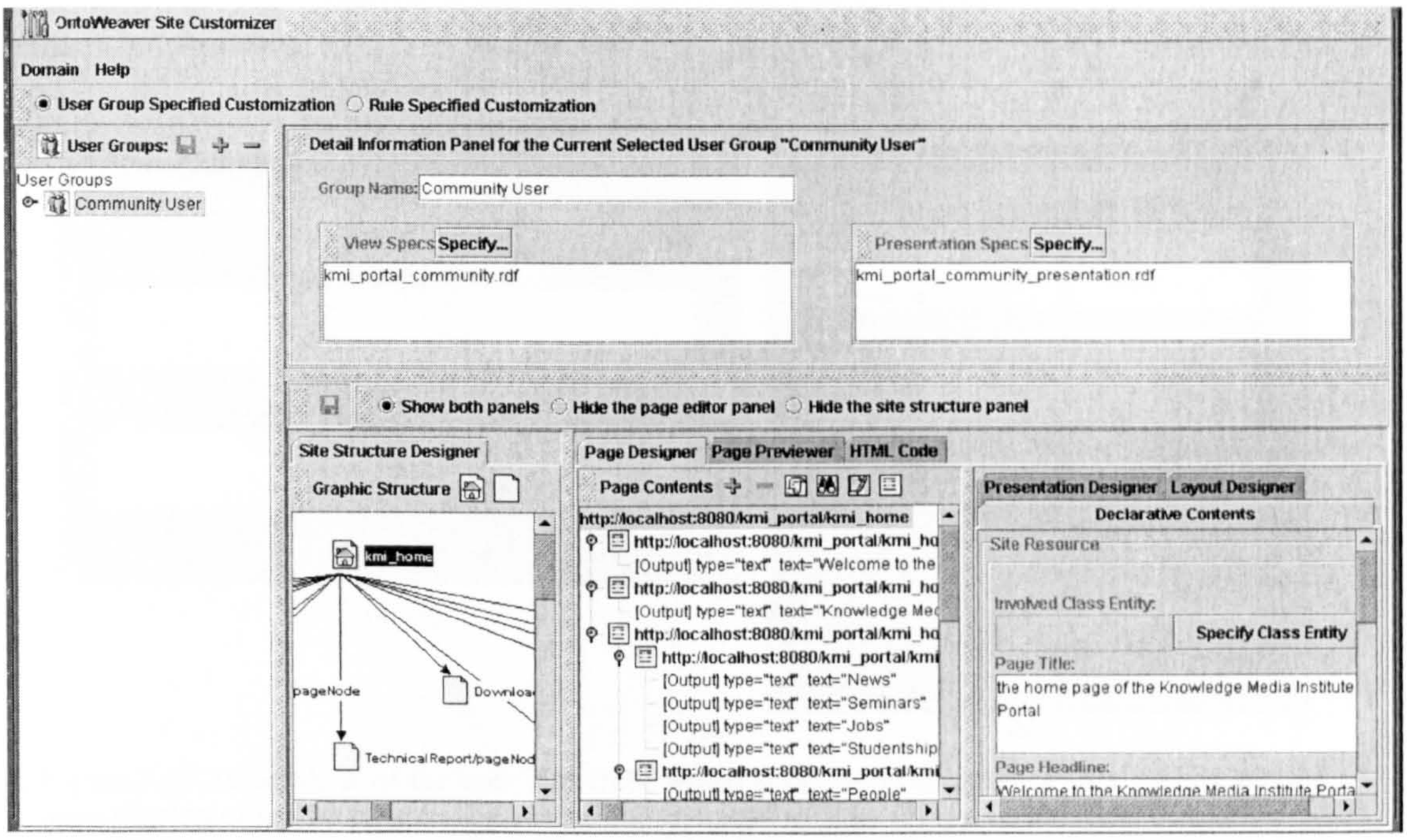


Figure 7.26 A screenshot of the user group customization pane which comprises three major components: i) *a user management pane* (located in the left side) supports the management of user groups and individual users, ii) *a site model assignment pane* (located in top of the right side) allows the specification of site models for the user group in question, and iii) *a site designer pane* (located in the bottom of the right side) facilitates the browsing and editing of the specified site models.

As described in chapter 6, OntoWeaver employs a class called *UserGroup* to describe user groups. Along with the creation of new user groups, instances of the

class *UserGroup* are created and stored in a domain specific Jess facts document, which is used to store user group information and user profiles. The user group information will be loaded into the customization engine to allow the retrieving of the user group specific site view model and site presentation models for end users.

Customizing site views for user groups

Now the site designer works on adapting the specified site views towards the group of *community users*: i) creating new web pages allowing the submission of news stories; and ii) adding links in other web pages, which allow the navigation to these web pages. These tasks can be achieved by means of the *site designer pane* which is located on the bottom of the right pane in figure 7.26. As this site designer pane is the same with the design pane contained in the OntoWeaver Site Designer, the process of editing user group specified site models correspond to the process of editing site models.

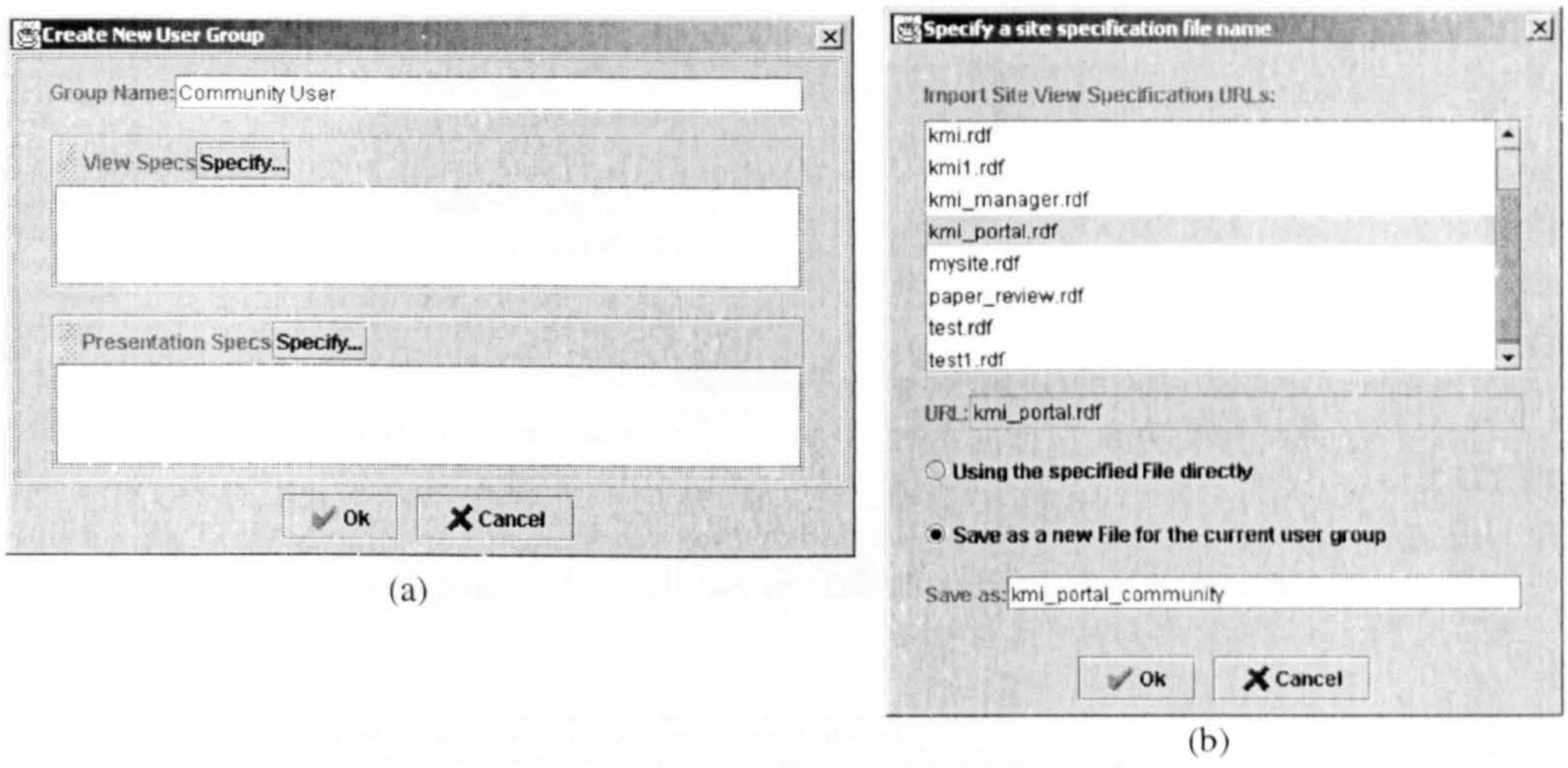


Figure 7.27 Screenshots of the user interfaces which support the creation of user groups and the specification of site models for the user group in question. Part (a) shows the form that allow for the definition of a user group. Part (b) shows the dialog that allows the specification of a site view model for the user group in question.

As explained in chapter 6, user group specific customization is realized by i) the OntoWeaver customization engine which finds the corresponding site models for the user in question (who has logged in the target web site) and applies the specified

customization rules (if there are any) to get customized models and ii) the Online Page Builder which generates customized web page for the end user.

Figure 7.28 shows the customized *seminar* web page for *Community User*s, who can submit the news of seminars to the web portal. A link has been added to the web page. Moreover, a new web page (as shown in figure 7.29) is created to allow the submission of Seminar news. These web pages are generated on the fly at run-time by *the OntoWeaver online page builder.*

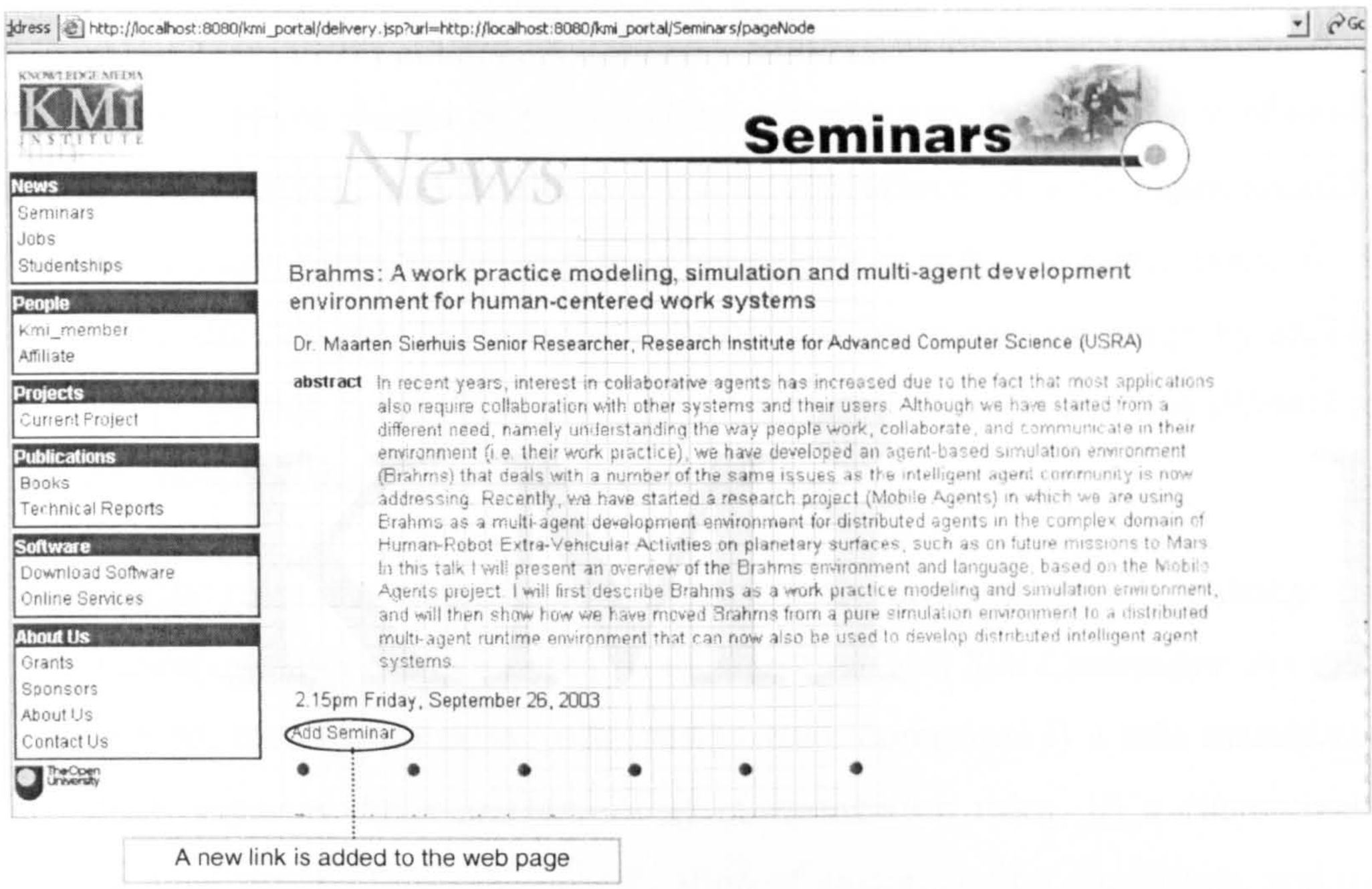


Figure 7.28 A screenshot of the customized Seminar web page for *community users*. A new link has been added to the web page, which allows the submission of new seminars.

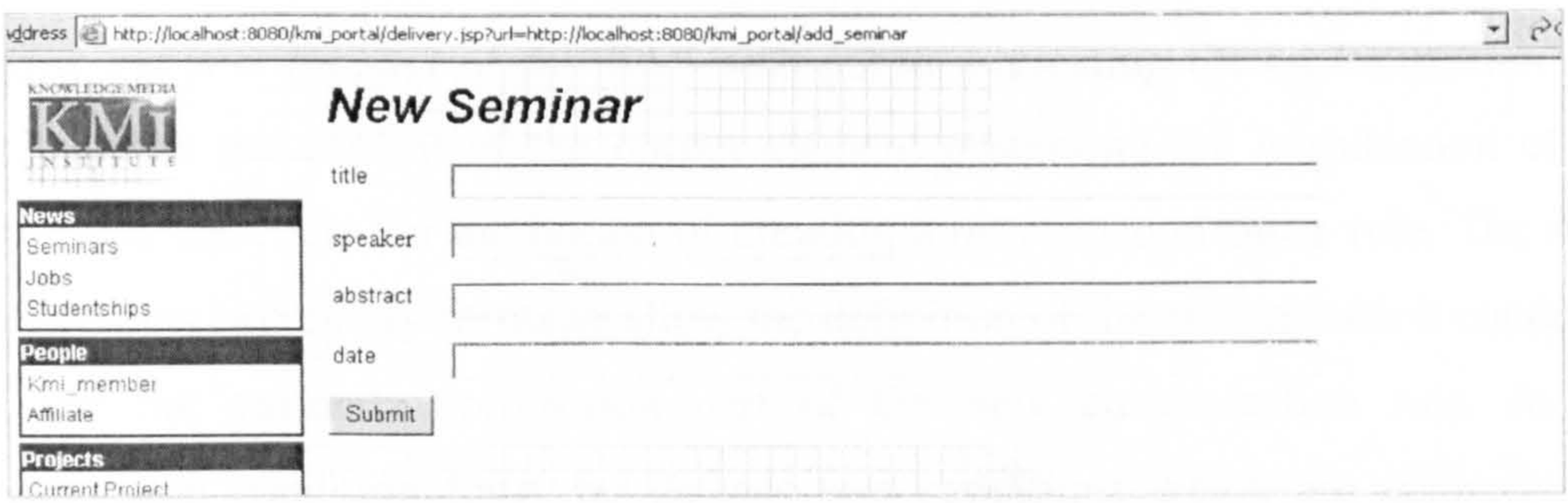


Figure 7.29 A screenshot of the web page which is generated allowing community users to submit new Seminar facts to the KMi web portal

7.2.5 User individual specific customization design

As described in chapter 6, user individual specific customization is based upon the specification of customization rules which defines when and how to perform customization. Such customization design in OntoWeaver involves i) building an appropriate user ontology which abstracts user information in a way that can reflect the individual requirements of users and ii) specifying customization rules. In the context of the KMi web portal, web pages should be personalized according to interest, preferences, and access devices of individual users. For example, web pages that present News stories should be tailored according to the interest of end users; the content of web pages should be personalized according to the screen size of devices that end users employ; and finally the visual appearance of web pages should be individualized according to the feature of user devices and user preferences. To this purpose, the site designer extends the OntoWeaver basic user ontology by adding a number of properties to the class *User*. For example, a property *interestedNewsTopic* is added to describe the topics that a user is interested in.

Customization rules can be specified by means of the customization rule design pane which is another major component of the design-time tool *Site Customizer*. As shown in figure 7.30, the customization rule design pane comprises i) a rule management pane which supports the management of customization rules, ii) a customization condition pane which allows the specification of customization conditions, and iii) a customization action design pane which supports the definition of customization actions.

Imagine the site designer of the KMi web portal is creating the customization rule which adapts the content of the output element presenting the introduction of the index page. He clicks on the button of creating a new customization rule. The right pane presents two empty forms to allow the definition of the customization condition part and the customization action part of the new customization rule. In the customization condition form, he defines two conditions which are connected to compose a complex condition by means of the logic operator "AND". In the customization action pane, he defines one customization action, which modifies the

value of the output element presenting the introduction of the index page as "This is created by OntoWeaver". Please note that more than one action can be defined in the customization action part.

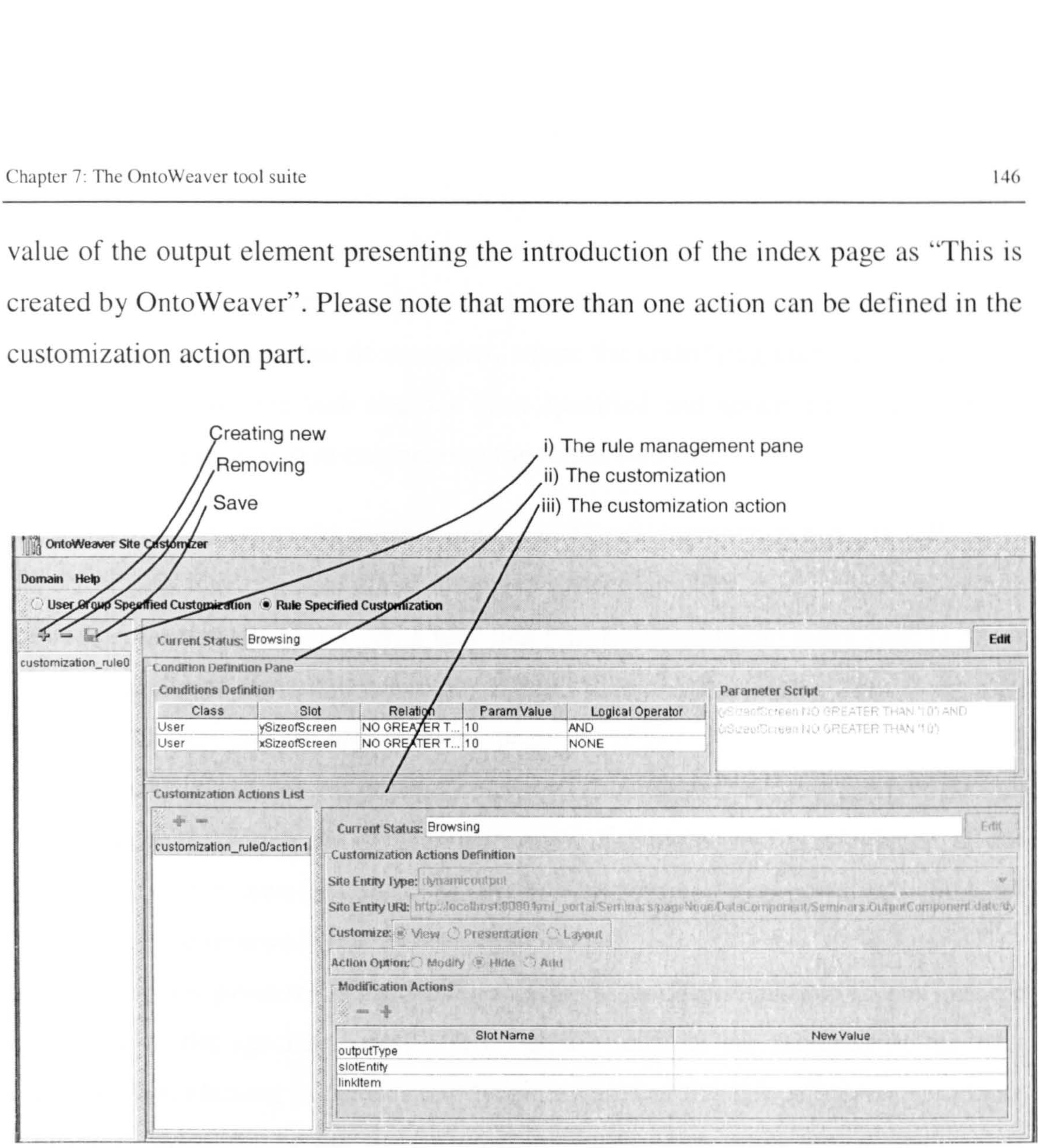


Figure 7.30 A screenshot of the customization rule design pane. The pane comprises i) *a rule management pane* which supports the management of customization rules, ii) *a customization condition pane* which allows the specification of customization conditions, and iii) *a customization action design pane* which supports the definition of customization actions.

7.3 Web site maintenance

Site maintenance is a key task in the life cycle of a web site. OntoWeaver supports this task in two ways. Firstly, the tool *Site Designer* allows the site views and presentations to be modified manually. In particular, the barrier to access and maintaining these specifications is very low, as the entire presentation layer and the site view layer are declaratively represented and can be modified with no need for programming. Secondly, the tool *Site Mapper* supports semi-automatic web site re-engineering after the domain ontology has been changed without loss of the

information made by developers during the site editing process, thus maintaining the conceptual links between site view models and the underlying domain ontology. Here we consider a number of scenarios, where the underlying domain ontology has been changed after the web site has been specified and generated, to explain the OntoWeaver approach to re-engineering the target web site.

- *Adding a new class to the domain ontology*. The re-engineering process will create a set of new web pages to allow the presentation, acquisition, and querying of this newly created class. Moreover, a new link will be added to the index page to allow the navigation to the data presentation page, which contains further links to the data acquisition page and the data querying page.
- *Removing a class from the domain ontology*. This requires the removing of the site view elements, which are dedicated to allow the access and manipulation of the instances of the specified class. Moreover, the relevant web pages and links are required to be removed from the site view model.
- *Adding a new property to an existing class*. This affects the web pages that are relevant to the specified class. New output elements are required in the data presentation element to present the dynamic value of the new property; new input elements are need in the data acquisition element to allow the acquisition of information about the new property.
- *Removing a property from a class*. Like the scenario described above, this affects the web pages that are relevant to the specified class. The corresponding dynamic output element and its explanation element should be removed from the data presentation element. Likewise, the input element of the specified property should be removed from the knowledge acquisition element.
- *Changing the definition of properties*. This is the composition of adding a new property and removing the old one. Thus, it can be dealt by using the methods described above.

In the KMi web portal, the domain expert changes the domain ontology by creating a new property *project_leader* and associating it with the class *Project*. This modification information is tracked in the Ontology Editor, which invokes the *Site Mapper* to maintain the conceptual relations between the domain ontology and the

corresponding site view specifications and invokes the Site Builder to re-generate the target web site subsequently. Figure 7.31 shows the screenshot of the project web page (which presents the instances of the class *Project*) after the re-engineering. A new component has been created for presenting values of the newly added slot. Moreover, the re-engineering does not lose any information designed by developers.

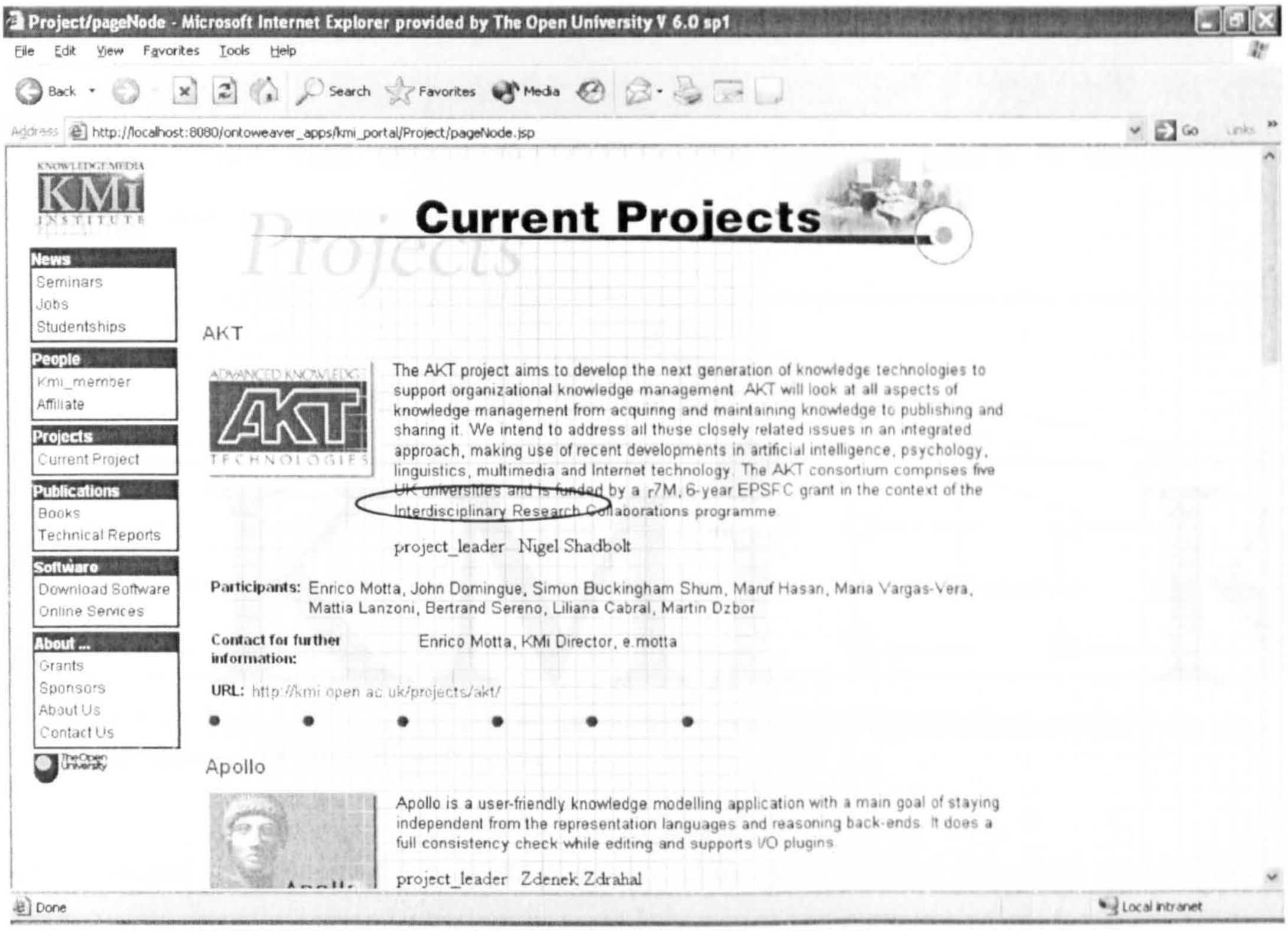


Figure 7.31 A screenshot of the project web page after web site re-engineering when a new slot called *project_leader* has been created for the class *Project*.

7.4 Automatic generation of web site specifications

The site view design process can be a completely manual process, in which web developers craft site views manually at the conceptual level according to particular requirements as described above, or an automatic or semi-automatic process, in which default site specifications can be produced by using the domain ontology to automatically instantiate the site view ontology, as the domain ontology and the site view ontology are both explicitly defined and each entity defined in the domain ontology is available for use from outside. Site implementations can then be

generated from the default site specifications (i.e. automatic site design) or from the manually tuned site specifications (i.e. semi-automatic site design).

The automatic generation of web site specifications starts from mapping the hierarchy structure of the domain ontology to a default site structure and navigation structure for the target web site. Each class, which needs to be instantiated at run time, is mapped to a set of page nodes. These page nodes include a page node for data presentation, a page node for data acquisition, and a page node for data querying. Figure 7.32 shows the default site structure which is automatically generated from the domain ontology of the KMi web portal.

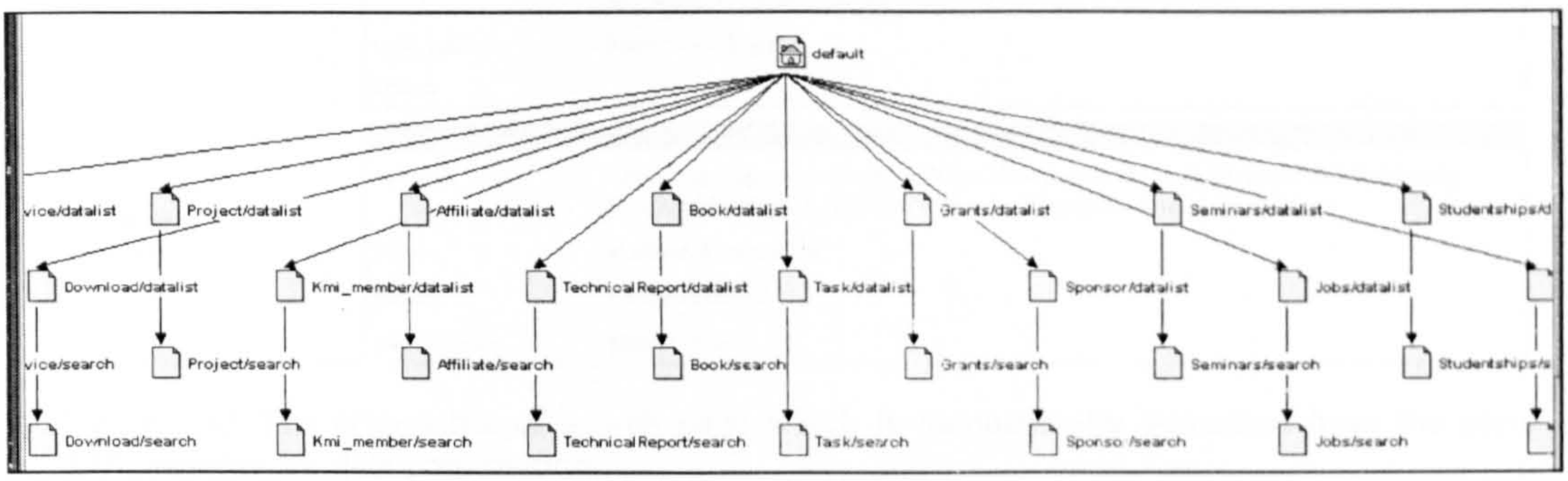


Figure 7.32 A screenshot of the default site structure which is automatically generated from the domain ontology of the KMi web portal

The process of generating a web page for data presentation involves generating a data component form the specified class entity. It scans the slots of the specified class and maps each slot to a static output element which presents an explanation about the value of the slot, and a dynamic output element which presents the dynamic value of the slot. Figure 7.33 shows a screenshot of an example web page, which is generated from the class *KMi_member*. The left component is the navigation structure generated from the class hierarchy structure of the domain ontology. The right component is the default data component generated from the definition of the class.

The processes of creating a knowledge-acquisition web page node and a data querying page node are similar to that of generating a data presentation page. Each slot of the specified class is mapped to an input element to allow users to provide information, and an output element to present an explanation about the input

element. The command elements contained in these page nodes are associated with built-in tasks which are *NEW-DATA-ENTRY* in knowledge acquisition page nodes and DATA-QUERYING in data querying page nodes.

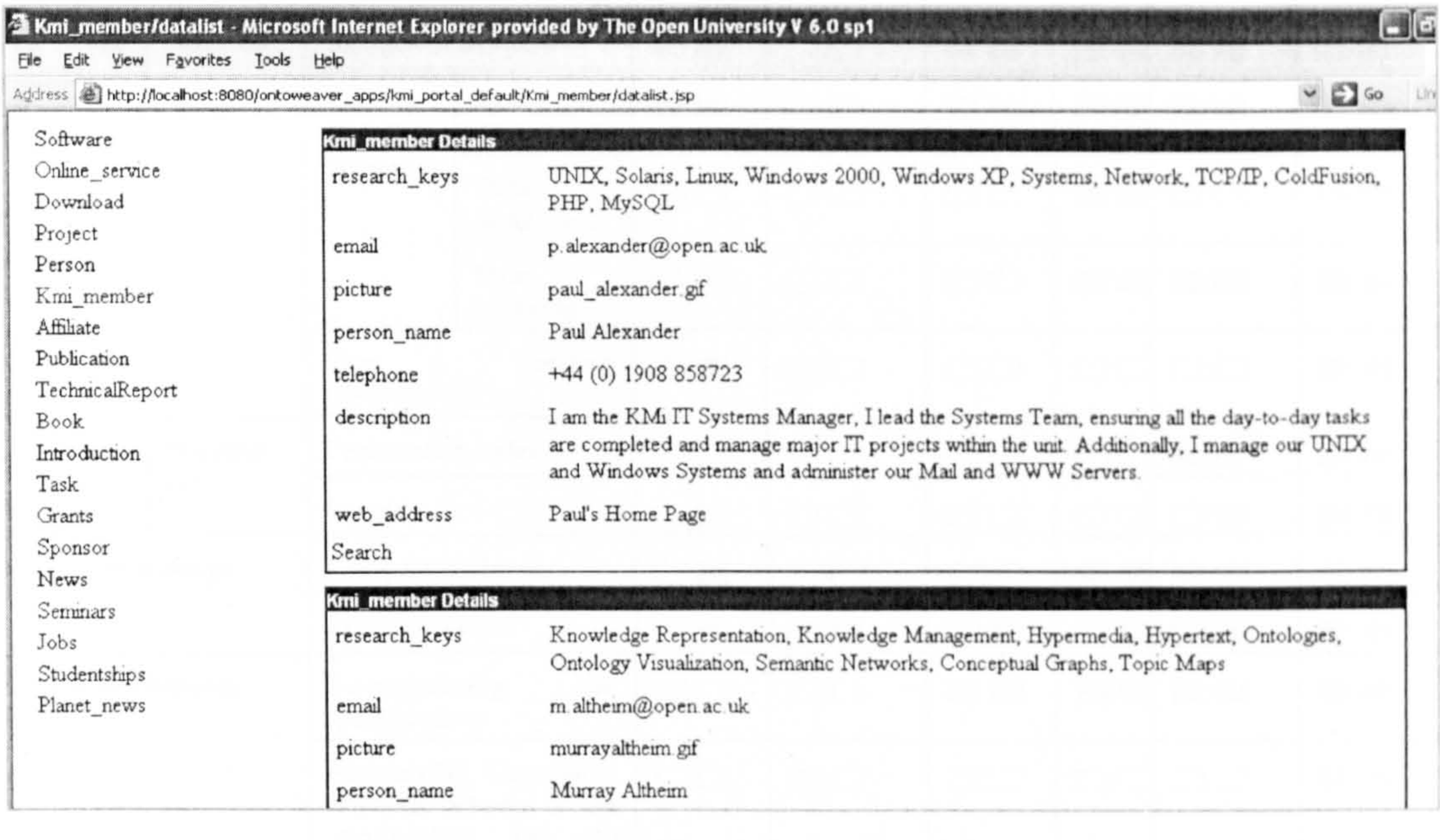


Figure 7.33 The screenshot of a web page which is automatically generated from the class *KMi_member* for data presentation.

7.5 A comparison

In chapter 5 and chapter 6, we have already related our work with conceptual web modelling approaches with respect to support for the specification of all aspects of data-intensive web sites, including navigation structures, user interfaces, presentation styles and layouts, and customization. In this section, we further compare OntoWeaver to current conceptual web modelling approaches with respect to support for the activities involved in the design and development of data-intensive web sites, including web site specification, maintenance, and customization. In particular, we compare OntoWeaver to the most recent approaches (including UWE, WebML, OntoWebber, HERA, and WUML) which are closer to OntoWeaver than the earlier approaches. Table 7.3 illustrate the comparison.

Table 7.3 A comparison between OntoWeaver and its closest approaches to web site specification, customization design, site design critiquing, and web site maintenance. We used three categories to classify support, including i) full support (■ ■), ii) partial support (□ ■), and iii) no support at all (□ □).

Dimension \ Tools				UWE	WUML	HERA	Web ML	Onto-Webber	Onto-Weaver
Web site specification	Site view	Navigation structure		■ ■	□ □	■ ■	■ ■	■ ■	■ ■
		Typical user interfaces	Data presentation	■ ■	□ □	■ ■	■ ■	■ ■	■ ■
			Data acquisition	□ □	□ □	□ □	■ ■	□ □	■ ■
			Data querying	□ □	□ □	□ □	■ ■	■ ■	■ ■
		User interface composition		□ ■	□ □	□ □	□ □	□ □	■ ■
	Presentation	Presentation styles		□ □	□ □	□ □	□ □	■ ■	■ ■
		Layout		□ □	□ □	□ □	□ □	□ ■	■ ■
Customization design		Composition level		□ ■	□ □	□ □	■ ■	■ ■	■ ■
		Derivation level		□ ■	□ □	□ ■	□ ■	□ ■	■ ■
Web site maintenance		Re-engineering site specifications		□ □	□ □	■ ■	■ ■	■ ■	■ ■
		Maintaining Conceptual relations between a site specification and the underlying domain model		□ □	□ □	□ □	□ □	□ □	■ ■
Site design critiquing				□ □	□ □	□ □	□ □	□ ■	□ ■

7.5.1 Support for the specification of web sites

As described in chapter 5, OntoWeaver provides comprehensive support for the specification of all aspects of data-intensive web sites, including navigation structures, user interfaces, presentation styles, and layouts, due to the explicit set of site ontologies. Thus, OntoWeaver overcomes the open issues associated with current approach, including i) limited support for the creation of complex site views and ii) limited support for the specification of presentation instructions.

Support for the specification of site views

As described in chapter 5, a site view of data-intensive web site comprises navigation structures which allow navigation from one page to another, and user interfaces which allow users to access information presented in web sites. The support for the

specification of site views thus comprises the support for the specification of navigation structures and the support for the specification of user interfaces.

As shown in table 7.3, the comparison on support for the specification of site views is carried out from three major dimensions, including the support for navigation structures, the support for typical user interfaces, and the support for user interface composition. All the approaches mentioned shown in table 7.3 except WUML (which dedicates to customization design) provide comprehensive support for the specification of navigation structures.

Typical user interfaces of data-intensive web sites allow end users to access (general users) and manipulate (advanced users) domain data stored in the underlying database. Like OntoWeaver, WebML provides comprehensive support for the specification of all typical user interfaces. OntoWebber and Hera only support some typical user interfaces: OntoWebber does not support the specification of user interfaces for data acquisition; and Hera also lacks the support for user interfaces which allow data querying.

User interface composition allows the specification of complex user interfaces using the provided constructs. This has not been addressed in most approaches, including HERA, WebML, and OntoWebber, due to the lack of fine-grained modelling support. UWE is an approach, which addresses the issue. It provides a set of *primitive* user interface elements, which present *texts*, *images*, *videos*, and *audios*. At the same time, UWE provides a set of composite user interface elements, which include *frameset*, *frame*, and *window*. However, UWE does not provide formal meta-model to support the specification of these constructs. Hence, it can only provide methodological guidance. OntoWeaver on the other hand provides comprehensive support by means of i) providing atomic user interface constructs, ii) allowing the assembling of atomic user interface elements as composite ones, and iii) providing explicit meta-models to support the specification.

Support for the specification of presentation styles and layouts

Most approaches do not take presentation styles and layouts into consideration. As a result, web developers need to rely on ad hoc approaches to realize the specification. OntoWebber is a partial exception, which provides constructs to describe presentation styles and a set of primitives to describe certain types of layouts. As described in chapter 2, the layout primitives can not support the specification of complex layouts. OntoWeaver provides comprehensive support for the specification of both presentation styles and layouts by means of its presentation ontology.

7.5.2 Support for customization design

As described in chapter 2, two solutions have been developed in current approaches to support the specialization of a general purpose of web site towards the profiles of user groups or individuals. They are the *composition-level* customization, which allows the construction of different site views at design time, and *derivation-level* customization, which allows the derivation of different site views at run time. Thus, we compare the support for customization design from these two levels.

Like OntoWeaver, most approaches support composition-level customization by allowing the creation of different site views for different user groups. HERA and WUML are exceptions which dedicated themselves to derivation-level customization support

Derivation-level customization is supported by all the approaches mentioned above. WebML and HERA rely on the annotation of web site specifications to realize customization. UWE and WUML rely on customization rules to specify customization requirements and derive customized site views. Although these approaches can provide comprehensive customization support, there are a number of limitations. Firstly, the scope of customization is limited, as not all aspects are specified declaratively and available for customization. Secondly, there is no support available for the specification of customization requirements. For example, no modelling support is available for the specification of annotation and the association of annotation with web site specifications in WebML and HERA and for the specification of rules in UWE and WUML. Finally, annotation-based approaches

(including WebML, HERA, and WUML) force web developers to anticipate what can be customized at the phase of site view design, which makes web modeling approaches not flexible.

OntoWeaver provides comprehensive customization support at design as well as run time. First, as all user interface elements and their presentation instructions are represented declaratively, the entire site model is available to customization. Second, as described in chapter 6, OntoWeaver relies on its customization framework to separate the specification of customization from other aspects of the target web site and provides specific support for the specification of customization requirements.

7.5.3 Support for site design critiquing

As described earlier, a set of built-in critiquing rules have been used in the OntoWeaver tool suite, which checks and presents basic errors of web site design, such as broken links and isolated web pages. Although this functionality is relatively simple at the moment, complex rules can be applied to provide more comprehensive critiquing services for web site design. Such facility is also supported in OntoWebber which makes use of a set of validation constraints to check the integrity of site specifications.

7.5.4 Support for web site maintenance

Site maintenance is a key task in the life cycle of a web site. Most approaches address this issue by allowing the modification of site specifications and the re-generation of web sites. This however can easily result in losing information which is not specified declaratively during web site design. As a consequence, without comprehensive meta-models, conceptual web modelling approaches can not provide appropriate support for web site maintenance. OntoWeaver addresses this issue by its comprehensive set of site ontologies.

As described earlier, the OntoWeaver tool suite also supports the maintenance of conceptual links between site specifications and the underlying domain data model,

thus making the target web site easier to maintain. Other tools on the other hand do not provide such facility. Hence, web sites constructed by these tools are brittle and hard to maintain.

7.6 Conclusions

In this chapter we have illustrated the facilities of the OntoWeaver tool suite by building a data-intensive web site example, the KMi web portal, which has been described in chapter 5. The illustration is carried out by using a number of scenarios to demonstrate how the OntoWeaver tool suite can be used to support different activities involved in the design of data-intensive web sites at a level which abstracts from low-level implementation issues. Specifically, we have explained the facilities that i) allow domain experts to create and edit domain ontologies, ii) support information architects, site designers, and layout designers in the specification of site structures, organization of information, composition of user interfaces, and definition of presentation styles and layouts, iii) facilitate customization design, and iv) support web site maintenance.

Furthermore, we have described the facilities that the OntoWeaver suite provides, which are powered by means of using site ontologies to represent web site specifications, including i) automatic site generation which produces default site specifications by means of using the domain ontology to instantiate the site view ontology, ii) semi-automatic site maintenance which supports the maintenance of conceptual links between site specifications and the underlying ontology, and iii) web site design critiquing which make use of a set of rules to assess the web site design result by reasoning upon the declarative site specifications and makes correction or improvement recommendations.

Finally, we have presented a comparison between the OntoWeaver approach and other conceptual web modelling approaches with respect to support for web site specification, customization design, web site design critiquing, and web site maintenance. The comparison shows that OntoWeaver provides more comprehensive support on these dimensions than current web modelling approaches.

Chapter 8: Integrating web services into data-intensive web sites

In this chapter we describe our work on extending OntoWeaver to OntoWeaver-S which aims to provide high-level support for the design and development of web service powered data-intensive web sites. We begin by describing the research motivation. We then present how to move OntoWeaver to OntoWeaver-S and how OntoWeaver-S supports the design of data-intensive web sites which allow access to both the underlying domain data and the specified web services. Next, we describe the implementation of OntoWeaver-S. Finally, we describe related work and reiterate the main contributions of this work.

8.1 Motivation

Designing web sites is a complex task. As discussed in chapter 2, a number of approaches have been developed to address this problem, in particular focusing on the design of data-intensive web sites. However, it is still the case that the functionalities of the target web sites are limited to the presentation and manipulation of the underlying data. Access to remotely available web applications (e.g., web services) is typically achieved in a time-consuming and expensive way by means of low-level programming. This functionality is however very important for web sites considering the benefits in terms of information sharing and information integration, deriving from access to remote web applications and remote data content.

An advantage of web services technology, compared to a generic provision of web applications, is that they allow access to remote content and application facilities, independently of specific implementations or data formats [Cabral et al., 2004] [Fensel & Bussler, 2002]. A number of mechanisms and platforms have been developed to support the specification, publication, discovery, and invocation of web

services [W3C, 2001] [OASIS, 2002] [W3C 2000b] [IBM, 2002] [SUN, 2003] [Ankolekar et al, 2002] [Fensel & Bussler, 2002] [Motta et al, 2003].

In this work we employ the technology of web services to address the issue of accessing remote web applications for data intensive web sites. We take our *data-driven* web site design framework, OntoWeaver, as a starting point and progress it to a *service-driven* web site design framework. We will call the extended architecture *OntoWeaver-S*.

8.2 From OntoWeaver to OntoWeaver-S

To bring web services to data-intensive web sites, we need to get a closer look at the technology of Web services and clarify the criteria required in a web site design framework to address the issue of integrating web services.

8.2.1 Web services

The key idea of the technology of web services is to use a standard protocol to allow applications to exchange data across the web. A web service is a software component which can be accessed via the internet through its exposed interface. The web service technology employs i) standardised registry mechanisms (e.g., in UDDI [OASIS, 2002]) to specify the locations of a web service, ii) web service description languages (e.g., WSDL [W3C, 2001], DAML-S [Ankolekar et al, 2002], and the Task-PSM model used in IRS-II) to describe the operations which can be performed by a web service and express how to invoke the web service, and iii) SOAP/XML [W3C, 2000b] to specify how to exchange messages across the web.

Figure 8.1 shows a usage scenario of accessing web services in a web application. Web service providers create their web services and publish them to a web service registry. Web applications as web service clients query the web service registry to obtain the definitions of the specified services and then invoke them according to the specifications.

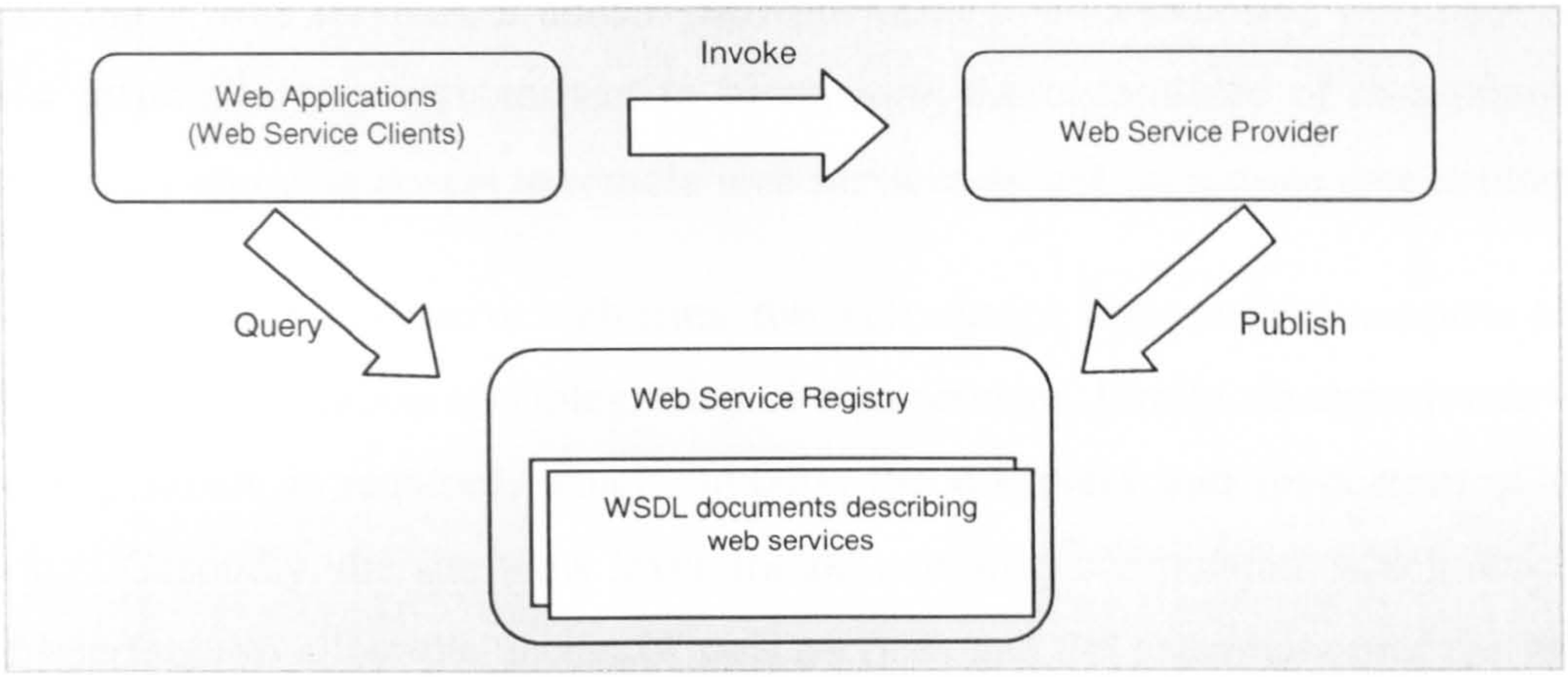


Figure 8.1 A usage scenario of accessing web services in web applications.

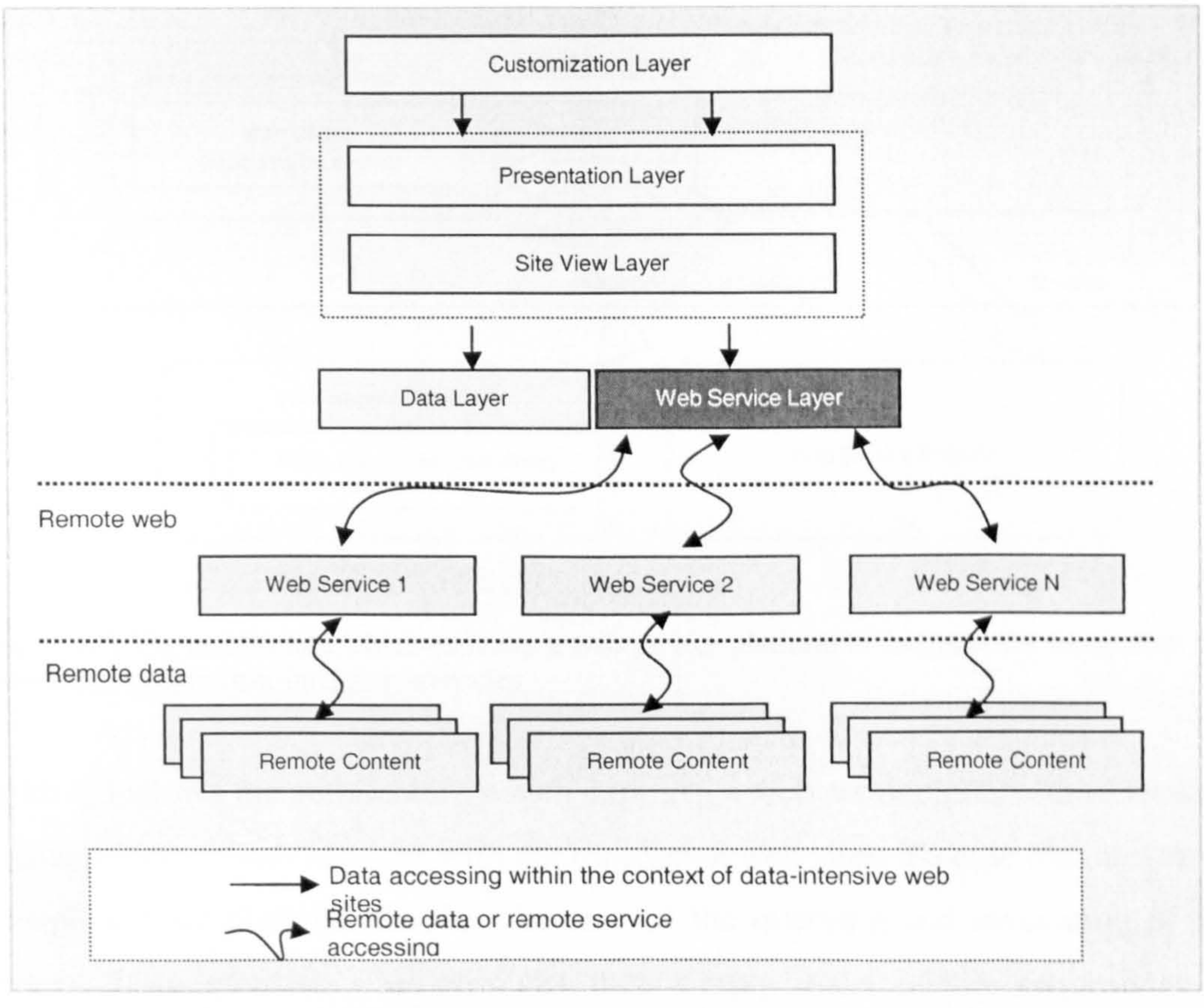


Figure 8.2 The extended architecture of data-intensive web sites

8.2.2 Integrating web services into data-intensive web sites

Figure 8.2 shows the extended architecture of data-intensive web sites which allow access to remote web services. A web service layer is added, which supports the

integration of web services. It adds significant value to data-intensive web sites, as it opens up possibilities with respect to broadening the capabilities of data-intensive web sites by allowing access to remote web services as well as remote data content.

To design such data-intensive web sites, the web service layer should comprise tools to allow the specification and integration of web services. Firstly, an appropriate web service platform is required, which supports the discovery and invocation of web services. Secondly, the site view layer should comprise components which describe user interfaces to allow the access of web services and the presentation of the result of web services.

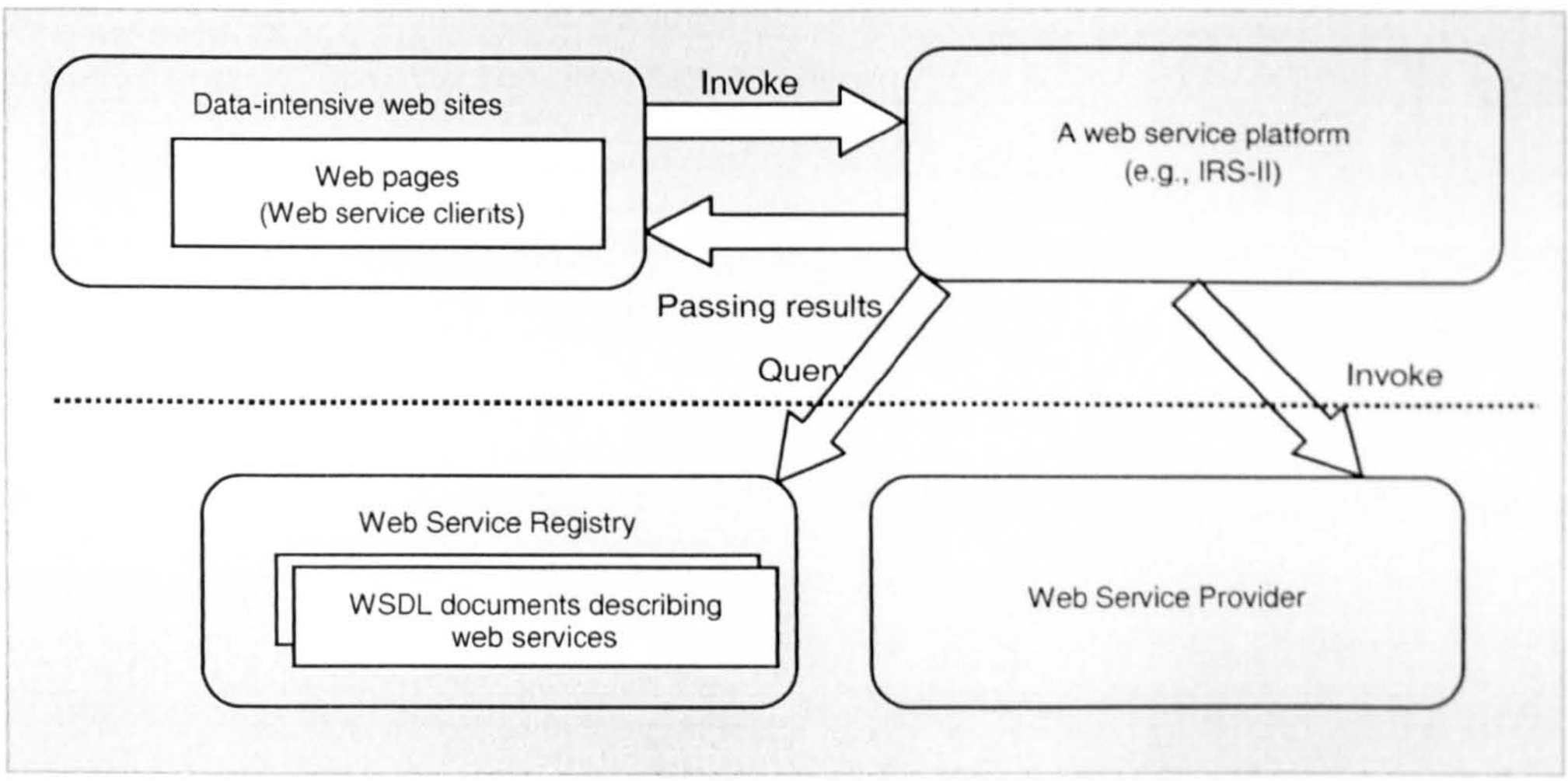


Figure 8.3 The architecture which explores a web service platform to facilitate the integration of web services with data-intensive web sites

Figure 8.3 shows the architecture which explores a web service platform to facilitate the integration of web services with data-intensive web sites. To ease the integration, the web service platform should take care of the querying and invocation of web services. Data-intensive web sites can then simply make service requests to the service platform and obtain the results of the execution of web services from the service platform. Moreover, these processes should be made as simple as possible for the integration.

In addition to an appropriate web service platform and a service integration tool, the web site design framework should provide site view constructs which support the

specification of user interfaces that allow access to web services. Furthermore, in order to allow web site developers to bring web services into the target web sites easily, tools should be provided to allow the specification of web services in the process of constructing site views at design time and to implement integration with web services at run time.

8.2.3 From OntoWeaver to OntoWeaver-S

As discussed in the earlier chapters, OntoWeaver employs the notion of ontology as the backbone to drive the design and development of data-intensive web sites. It provides the site view ontology and the presentation ontology to enable the declarative representation of data-intensive web sites. Moreover, it includes a generic customization framework to offer high level support for the specification of customization requirements at design time and to offer comprehensive customization support at run time. Finally, it offers a tool infrastructure to support the design and development of data-intensive web sites. However, like other web modelling approaches, OntoWeaver focuses on data presentation and does not offer high level support for bringing web services into web sites.

OntoWeaver-S is an OntoWeaver extension, which supports the design of web sites that are powered by web services as well as the underlying domain data. In moving from OntoWeaver to OntoWeaver-S we have to introduce a number of changes according to the criteria we elaborated in the previous section:

- Extending the site view ontology to allow the high level specification of web services within the site view elements of target web sites.
- Implementing tools to support the design and development of service centred web sites. In particular, the *Site Designer*, which supports the design and development of data-intensive web sites in OntoWeaver, should be extended to offer support for the specification of web services in the target web sites. At the same time, a new run time tool, called *Service Integrator*, needs to be built for integrating web services into web sites.

IRS-II as the web service platform

A number of web service platforms have been developed, which support the publication, discovery, and invocation of web services. Examples include the Java Web Services Developer Pack (JWSDP) [SUN, 2003], the IBM Web Services Toolkit (WSTK) [IBM, 2002], and IRS-II [Motta et al., 2003]. JWSDP and WSTK provide powerful programming environments to allow the creation, publication, and invocation of web services and the building of web applications based on web services. IRS-II is a platform to support the handling of semantic web services which are web services with semantic descriptions.

As discussed earlier, an appropriate web service platform is required to ease the integration of web services with data-intensive web sites. The criteria we used to choose a web service platform are i) the platform should be fully implemented, ii) the platform should provide comprehensive support for the discovery and invocation of web services, and iii) it should be easy to integrate within web applications. Considering these criteria, we choose IRS-II as the platform to do our experiment in this work. IRS-II is a fully implemented infrastructure, which focuses on the publication, discovery, and execution of semantic web services. It hides most of the programming part from users (i.e. web developers) which is typically required in other web service platforms (e.g. JWSDP and WSTK) thus providing an easy interface supporting the specification and invocation of web services.

Semantic web services augment web services with rich formal descriptions of their functionalities [Ankolekar et al, 2002] [Fensel & Bussler, 2002] [Motta et al, 2003]. The following informal specification shows the semantic description of an IRS-II semantic web service, which answers requests for flights in accordance with the given user requirements.

```
Task Ontology: flight-service
Task Name:  find-flights
Input Roles:  from-place (type: city)
              to-place (type: city)
              depart-time (type: time-point)
              arrival-time (type: time-point)
              budget (type: amount-of-money)
Output Roles: flights (type: Flight)
```

To invoke this semantic web service, a user simply asks for the task to be achieved in terms of the task name *find-flights* and the task ontology name *flight-service*, the IRS-II *broker* then selects an appropriate problem solving method (PSM) to locate and invoke the corresponding web service – see [Motta et al., 2003] for a detailed description of IRS-II. In particular, the *input roles* carry parameters for executing the corresponding web service; the *output roles* store the service results. However, IRS-II currently only allows one output role. The data type of the output role can be either a primitive type (e.g., string and integer) or a non-primitive one (e.g., domain classes). When the data type of the output role is not primitive, IRS-II uses XML to represent the service results. For example, IRS-II uses XML to represent the results of the web service find-flights, which are instances of the class *Flight*. This class has been defined in the domain ontology of the semantic web service.

Figure 8.4 shows the process of accessing remote web services in an OntoWeaver-S generated data-intensive web site. OntoWeaver-S provides a run-time tool called *Service Integrator* to integrate IRS-II with data-intensive web sites. Specifically, the Service Integrator collects information from an OntoWeaver-S generated web site, then calls the IRS-II server (by means of the IRS-II APIs) to invoke the specified web service, gets the results from IRS-II, and finally passes the results back to the web site. This process is completely transparent and web developers only need concern themselves with the functionalities of they interest to achieve.

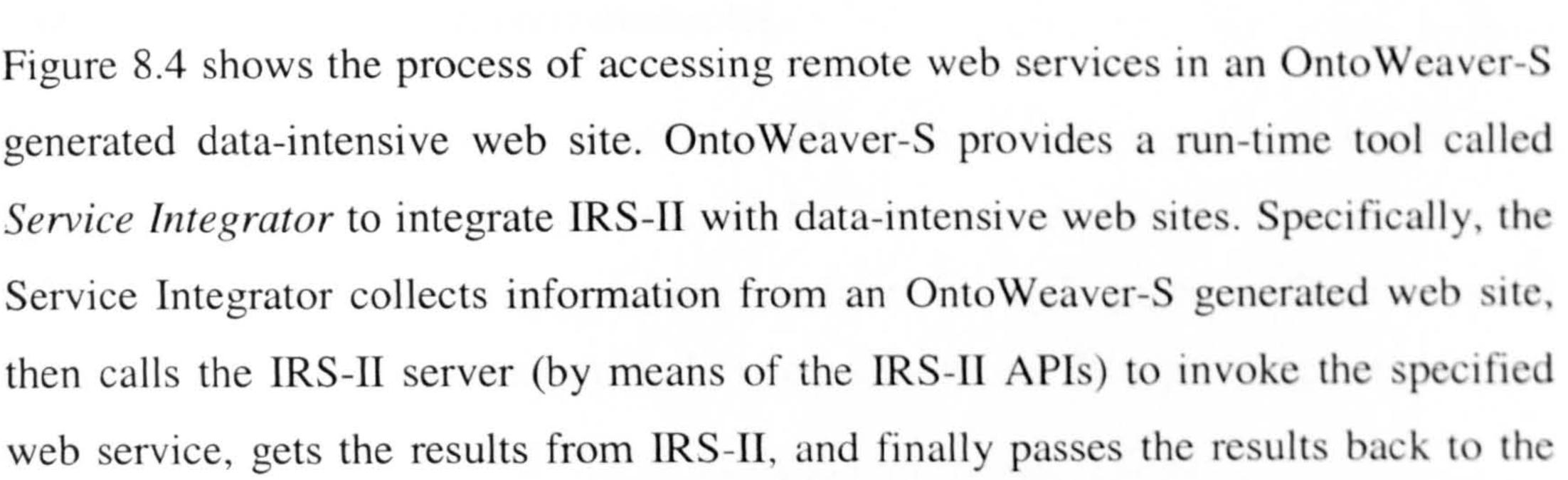


Figure 8.4 The process of accessing web services in an OntoWeaver-S generated web site

Web site design process in OntoWeaver-S

The web site design process in OntoWeaver-S comprises the following steps:

- *Requirements collection and analysis* identify the objectives of the target web sites, data characteristics, user categories, user requirements, and so on.
- *Domain ontology design* develops a data model to abstract the problem domain.
- *Web service collection* identifies and gathers the web services that are needed in the target web site.
- *Site structure design* defines the site structure for the target web site.
- *Page content design* produces detailed content for each page node defined in the site structure. In particular, the user interfaces for accessing web services are specified at this stage.
- *Presentation design* specifies presentation styles and layouts for site view elements in the site view model.
- *Customization design* identifies and specifies customization requirements in terms of user ontology and customization rules.

Comparing to OntoWeaver, OntoWeaver-S adds a web service collection and specification step to the design process. Furthermore, as will be described in the following section, the design of user interfaces for web pages in OntoWeaver-S involves not only data presentation, data acquisition, and data querying but also user interfaces for web service access.

8.3 Modelling user interfaces for accessing web services

To allow the specification of web services in the site views of data-intensive web sites, we have to extend the site view ontology in OntoWeaver-S. Within the extended site view constructs, remote web services are described in terms of *tasks*, *input roles*, and *output roles*, consistently with the semantic representation approach employed in IRS-II.

8.3.1 Information provision

Information provision is realized through knowledge acquisition forms which allow users to submit information to web sites. The submitted information can be records of the underlying databases or information on how to invoke the specified service.

Three constructs have been defined in OntoWeaver to allow the specification of user interfaces of knowledge acquisition. They are *KAComponent*, *Input*, and *Command*:

- The construct *KAComponent* models the user interfaces of the knowledge acquisition forms in which end users can fill information and create new facts by submitting the required information.
- The construct *Input* describes the actual input fields which allow end users to specify information for particular slots of the specified domain class entity.
- The construct *Command* describes the user interface elements which allow end users to trigger the associated services.

These constructs have been extended in OntoWeaver-S in order to describe user interfaces enabling information provision for web services. In particular, the construct *KAComponent* has been extended by adding a property *hasTask* to describe the associated task. The construct *Input* has been extended by adding a property *hasTask* and a property *hasInputRole* to describe the corresponding input role in the associated task. The construct *Command* has been extended by adding a property *hasTask* to describe the associated task.

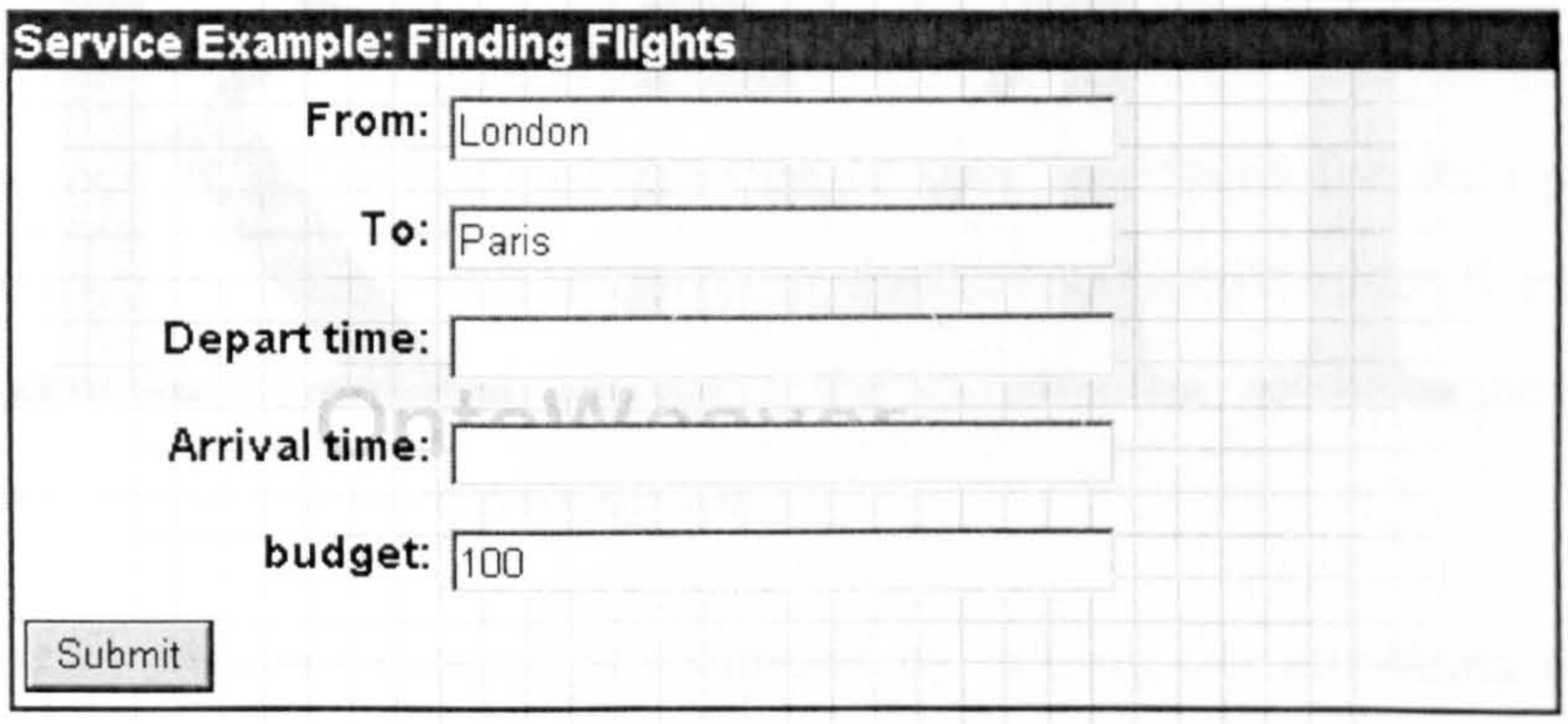


Figure 8.5 A sample user interface for accessing the web service find-flights

Figure 8.5 shows a sample user interface of the KA component for accessing the remote web service *find-flights*. The user interface is made up of a number of input fields and a command button for allowing end users to invoke the web service. The following shows a fragment of specification of this knowledge acquisition component.

```
<rdf:Description about="find-flights-page/kacomponent" >
   <rdf:type rdf:resource="&svo;KAComponent"/>
   <svo:task rdf:resource="find-flights"/>
   <svo:inputComponent>
     <rdf:Bag>
      <rdf:li resource="find-flights-page/kacomponent/from-place"/>
      <rdf:li resource=" find-flights-page/kacomponent/to-place"/>
      ....
     </rdf:Bag>
   </svo:inputComponent>
  <svo:command rdf:resource="kacomponent/command" />
</rdf:Description>

<! -- an input component example, which is composed of a static output element and an
input element -->
<rdf:Description about="find-flights-page/kacomponent/from-place" >
  <rdf:type rdf:resource="&svo;InputComponent"/>
     <svo:output resource="kacomponet/from-place/output"/>
     <svo:input resource="kacomponet/from-place/input"/>
</rdf:Description>
...
<! -- an input element example, which allows the typing of information for the input
role "from-place" -->
<rdf:Description about="kacomponent/from-place/input" >
     <rdf:type rdf:resource="&svo;Input" />
     <svo:task rdf:resource="find-flights"/>
     <svo:inputRole rdf:resource="find-flights/param/from-place"/>
</rdf:Description>

<!--  the definition of the command element -->
<rdf:Description rdf:about="kacomponent/command">
   <rdf:type rdf:resource="&svo;Command"/>
   <svo:commandText>Submit</svo:commandText>
   <svo:task rdf:resource="find-flights"/>
   <svo:resultPage rdf:resource="flights-result-page" />
</rdf:Description>
```

8.3.2 Data presentation

As discussed in chapter 5, OntoWeaver provides the constructs *DynamicOutput* and *DataComponent* to allow the specification of user interfaces for data presentation, with the data coming from the underlying databases. OntoWeaver-S extends these components by adding properties to allow the specification of the associated service and the fields whose values are to appear:

- The construct *DynamicOutput* is extended by adding the attributes *hasTask* and *hasOutputRole* to describe the associated task and the output field. The following RDF code defines one dynamic output element for publishing the dynamic content of the field *airline* which is the result of the task *find-flights*.

```
<rdf:Description about="datacomponet/dynamicoutput/airline" >
   <rdf:type rdf:resource="&svo;DynamicOutput" />
   <svo:outputType>text</so:outputType>
   <svo:task rdf:resource="find-flights" />
   <svo:outputRole rdf:resource="find-flights/airline" />
</rdf:Descripltion>
```

- The construct *DataComponent* is extended by adding the attribute *hasTask* to specify the associated task. Figure 8.6 shows a sample user interface for visualizing the results of the web service *find-flights*. This user interface contains a number of dynamic output elements for visualizing the output values of the web service. The following code illustrates the composition of this data component.

```
  <!--  the composition of the entire data component -->
 <rdf:Description about="flights-result-page/datacomponent" >
  <rdf:type rdf:resource="&svo;DataComponent"/>
  <svo:task rdf:resource="find-flights"/>
  <svo:outputComponent>
  <rdf:Bag>
    <rdf:li resource="flights-result-page/datacomponent/airline"/>
    <rdf:li resource="flights-result-page/datacomponent/fromairport"/>
     ……
  </rdf:Bag>
  </svo:outputComponent>
</rdf:Description>

 <!-- the composition of an output component    -->
<rdf:Description about="flights-result-page/datacomponent/airline" >
  <rdf:type rdf:resource="&svo;OutputComponent"/>
   …
   <svo:dynamicOutput rdf:resource= "datacomponet/dynamicoutput/airline" />
</rdf:Description>
…
```

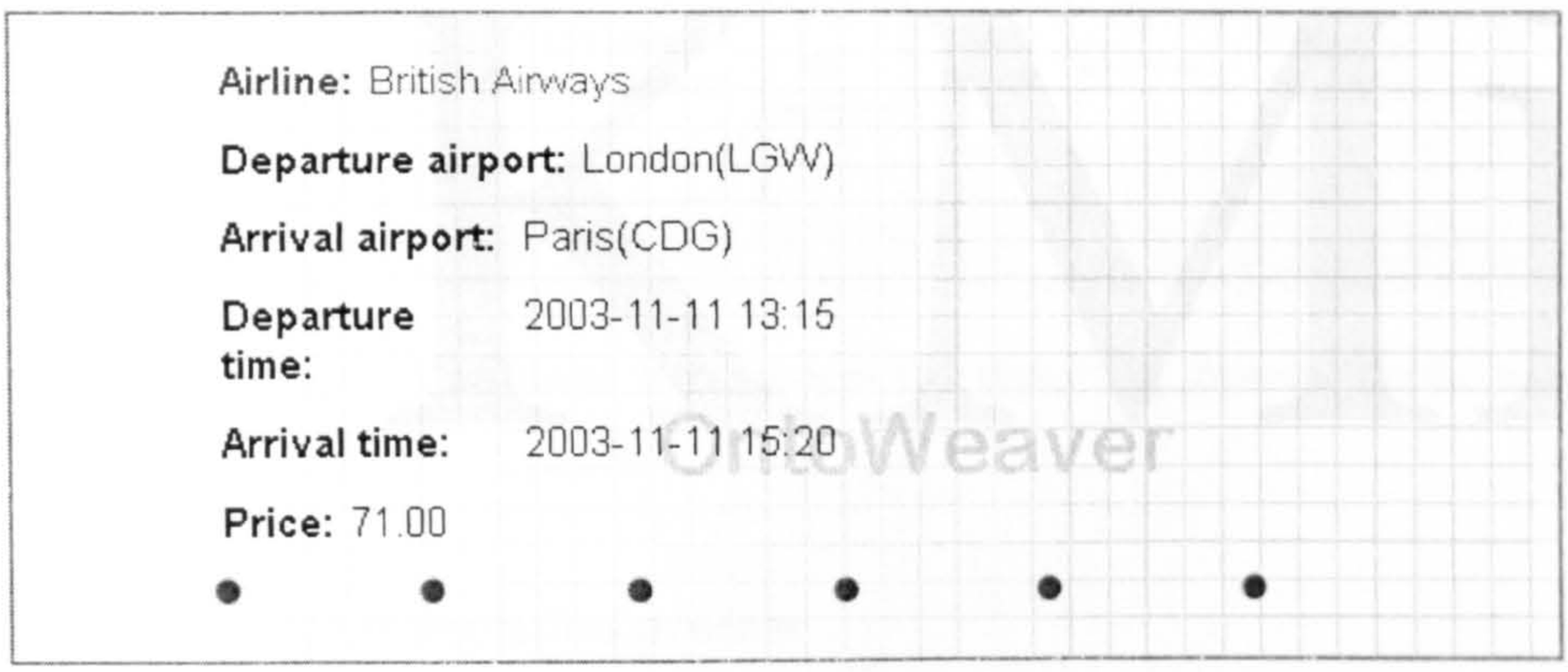


Figure 8.6 A sample user interface for presenting the results of the web service *find-flights*

8.4 Implementation of the web service integration

In this section, we build a simple web site for accessing the example web service *find-flights* to illustrate the implementation of the web service integration. In particular, we create two web pages according to the following steps (please note that as will be described later, we adapted the OntoWeaver Site Designer to support the design and development of service centred data-intensive web sites):

- Creating a page node called *find-flights-page* for holding components that allow end users to find flights according to their requirements.
- Creating *a knowledge acquisition component* in the page node *find-flights-page*, specifying the associated web service as the web service find-flights and choosing appropriate input roles.
- Creating a page node called *flights-result-page* for publishing the service result.
- Creating a data component in the page node *flights-result-page*, associating it with the web service *find-flights* and choosing appropriate output roles for the publication of service results.
- Specifying the page node *flights-result-page* as the value for the attribute *hasAssociatedResourceURI* of the command element contained in the knowledge acquisition component.

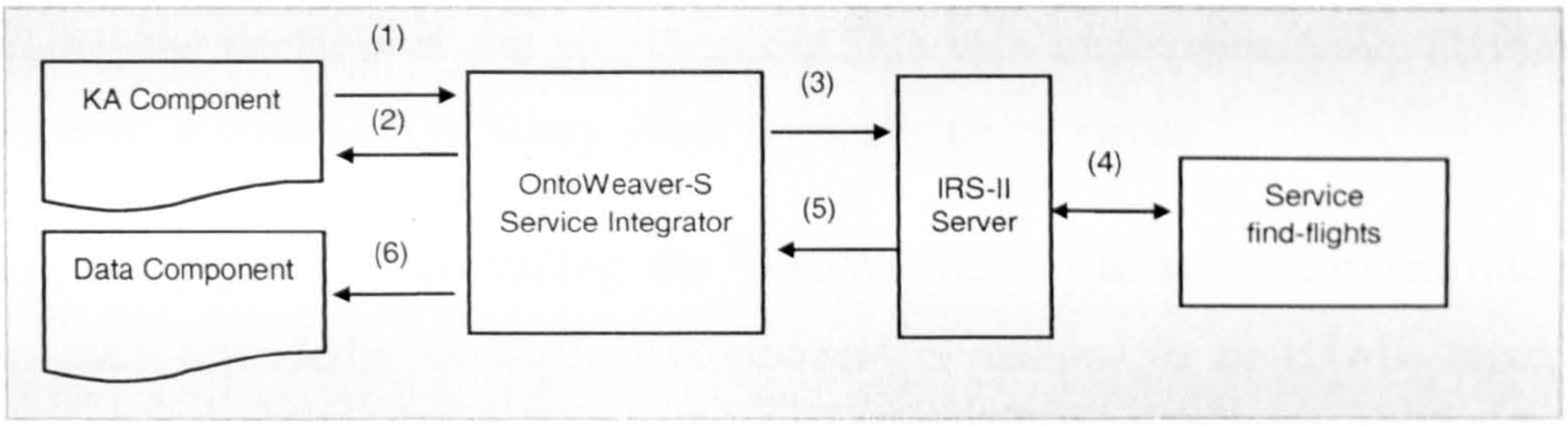


Figure 8.7 The process of accessing web services and presenting dynamic results coming from web services.

8.4.1 The web service integration process

In this section, we get a closer look at the process of accessing web services in the OntoWeaver-S generated data-intensive web sites. As illustrated in figure 8.7, the process of accessing web services comprises the following steps:

(1) An end user accesses the web page *find-flights-page*, enters his or her requirements for finding flights and submits the input information to the web site.
(2) The OntoWeaver-S Service Integrator investigates the input elements, which are contained in the web page making a request for accessing the specified web service, and gathers information from the input elements for the corresponding input roles of the specified web service.

(3) The OntoWeaver-S Service Integrator calls the IRS-II Server to achieve the specified task find-flights augmented with the given constraints.

(4) The IRS-II Server invokes the corresponding web service.

(5) The IRS-II Server returns the results of the execution of web services to the OntoWeaver-S Service Integrator.

(6) The OntoWeaver-S Service Integrator passes the results back to the web page flights-result-page.

8.4.2 Implementation

In this section, we discuss the implementation issues of the integration of semantic web services into web sites. It should be noted that the code described in this section is generated automatically by the OntoWeaver-S Site Builder during the process of compiling the declarative site specifications into web implementations. Developers do not need to cope with or worry about implementation issues.

During the process of compiling the OntoWeaver-S site specifications into web pages, each knowledge acquisition component is mapped to an HTML form. The command element contained within the component is mapped to a submit button to enable the submission of input information and the invocation of the specified semantic web service. To enable readability, the following HTML code represents a simplified JSP form, which is generated from the knowledge acquisition component example discussed earlier. The *action attribute* of the form is generated from the command element to indicate that the processing page for the form submission is *flights-result-page.jsp*. The *input fields* are created from the input elements.

```
<form action="flights-result-page.jsp" method="POST">
   <INPUT TYPE="text" name="kacomponent/from-place/input" >
   <INPUT TYPE="text" name="kacomponent/to-place/input" >
  ......
</form>
```

Each data component is mapped to an HTML table to present dynamic content coming from the associated semantic web service. Moreover, additional server-side code is generated at the same time to enable the access of the specified web service and the publication of the service results. The following code shows the simplified

JSP code generated from the definition of the data component example illustrated earlier. Please note that the sign of "<%" and "%>" indicates that the wrapped code is server-side code, which is executed by the web server at run time.

```
 <!– Part I: instantiating a Service Integrator -- >
<jsp:useBean id="function_find_flights"
        scope="session" class="ontoweaverbean.ServiceIntegrator"/>
<jsp:setProperty  name="function_find_flights"
        property="taskName" value="find-flights" />
<jsp:setProperty name="web_site_query"
        property="taskOntologyName" value="flight-service"/>

<% //Part II: adding input roles for the specified task
function_find_flights.addInputRole("from-place",
                                request.getParameter("kacomponent/from-place/input");
function_find_flights.addInputRole("to-place",
                request.getParameter("kacomponent/to-place/input");
……

 // Part III: achieving the specified task
function_find_flights.achieveTask(); %>

<!–part IV: presenting results of the web service  -->
<%while (function_find_flights.hasNextResultOutput())
      { %>
  <table>
     <tr><td> <%=function_find_flights.get("airline") %> </td></tr>
     <tr><td><%=function_find_flights.get("fromairport")%></td></tr>
        ……
  </table>
<% }%>
```

The code comprises four parts: i) the first part instantiates a service integrator using the specified task name and the task ontology name according to the specification of the associated semantic web service, ii) the second part gathers input information from the corresponding input elements and passes the information to the corresponding input roles according to the specification of the corresponding knowledge acquisition component, iii) the third part achieves the specified task by calling the IRS-II Server, and iv) the fourth part presents the service results in a table.

The following Java code shows how the Service Integrator achieves its goal of integrating web services. First, it calls the function *achieveTask()* from the IRS-II Server, augmenting the specified task name, task ontology name, and the values for the input roles. The result of this function is written in XML (this has been indicated by the semantic description of this web service). The OntoWeaver-S Service Integration then gets the result from the IRS-II Server and processes the result, and makes it ready for the presentation.

```
public void achieveTask()
{
```

```
    String serviceResult =
            this.irsServer.achieveTask(this.taskName,this.taskOntologyName,
                                       inputtRoles);
    this.processingResultOutputs(serviceResult);
}

//processing results and making it ready for publication
private void processResultOutputs(String serviceResult)
{
    ResultOutputReader reader=new ResultOutputReader(serviceResult);
    this.resultOutputList=reader.getResultOutputList();
 }

public boolean hasNextResult()
{
   boolean hasNext= this.resultOutputList.hasNext();
   if (hasNext)
        this.currentRowResult= this.resultOutputList.next();
   return hasNext;
}

//getting the value of the specified result field of the current row
public String get(String outputName)
{
    return this.currentRowResult.get(outputName);
}
```

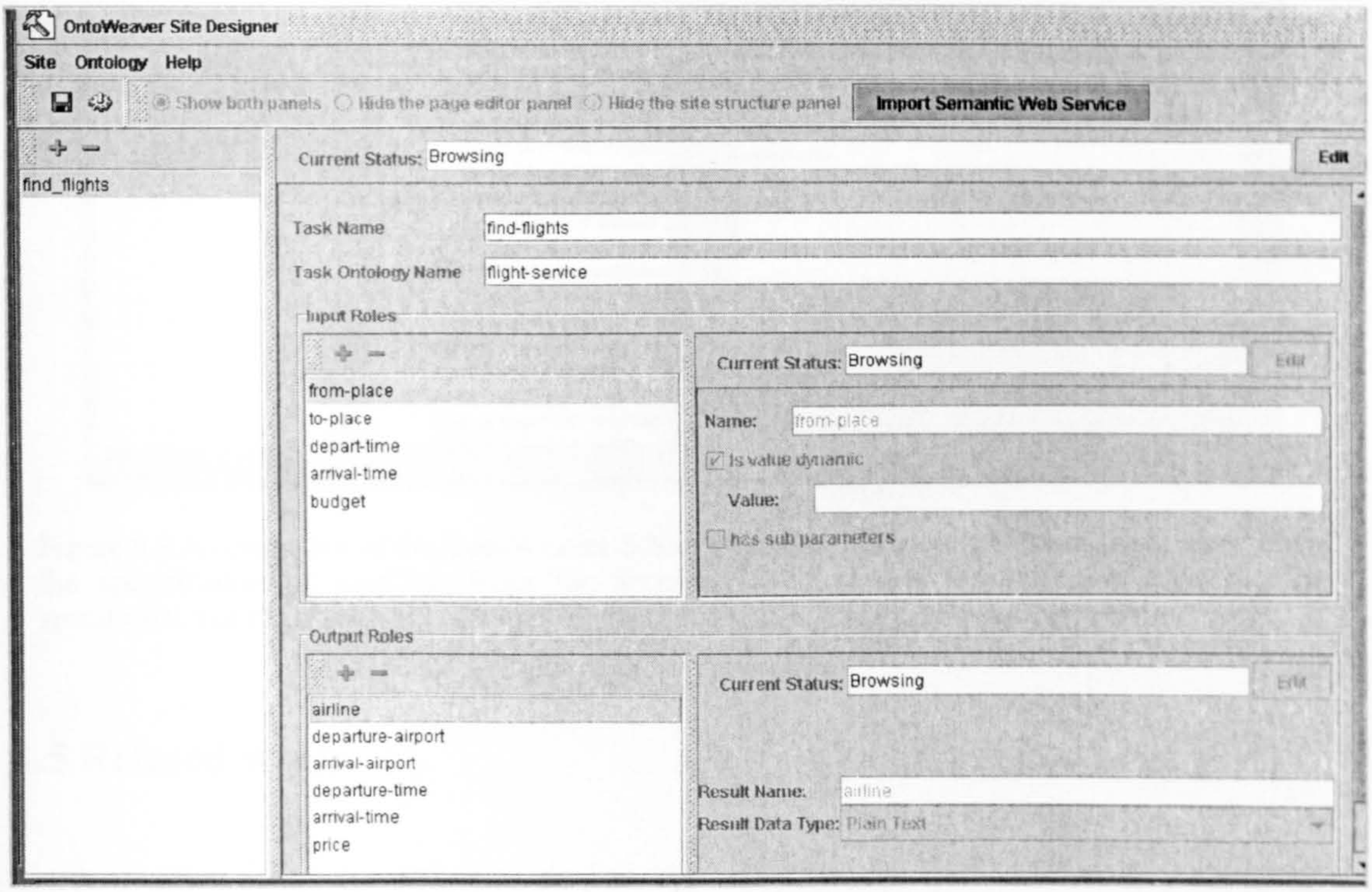


Figure 8.8 A screenshot of the Web Service Importer which is a new component of the tool Site Designer. The Web Service Importer facilitates the specification and management of semantic web services which will be integrated with the target web site. The integration is achieved when developers create user interfaces for accessing the specified web services.

In addition to the provision of the tool *Service Integrator*, we also adapted the OntoWeaver Site Designer to support the design and development of service centred data-intensive web sites. Figure 8.8 shows a screenshot of the *Web Service Importer*

which is a new component of the tool *Site Designer*. The Web Service Importer facilitates the specification and management of semantic web services which will be integrated with the target web site. The integration is achieved when developers create user interfaces for accessing the specified web services. Figure 8.9 shows a screenshot of the declarative content pane which facilitates the definition of site view elements for web service access.

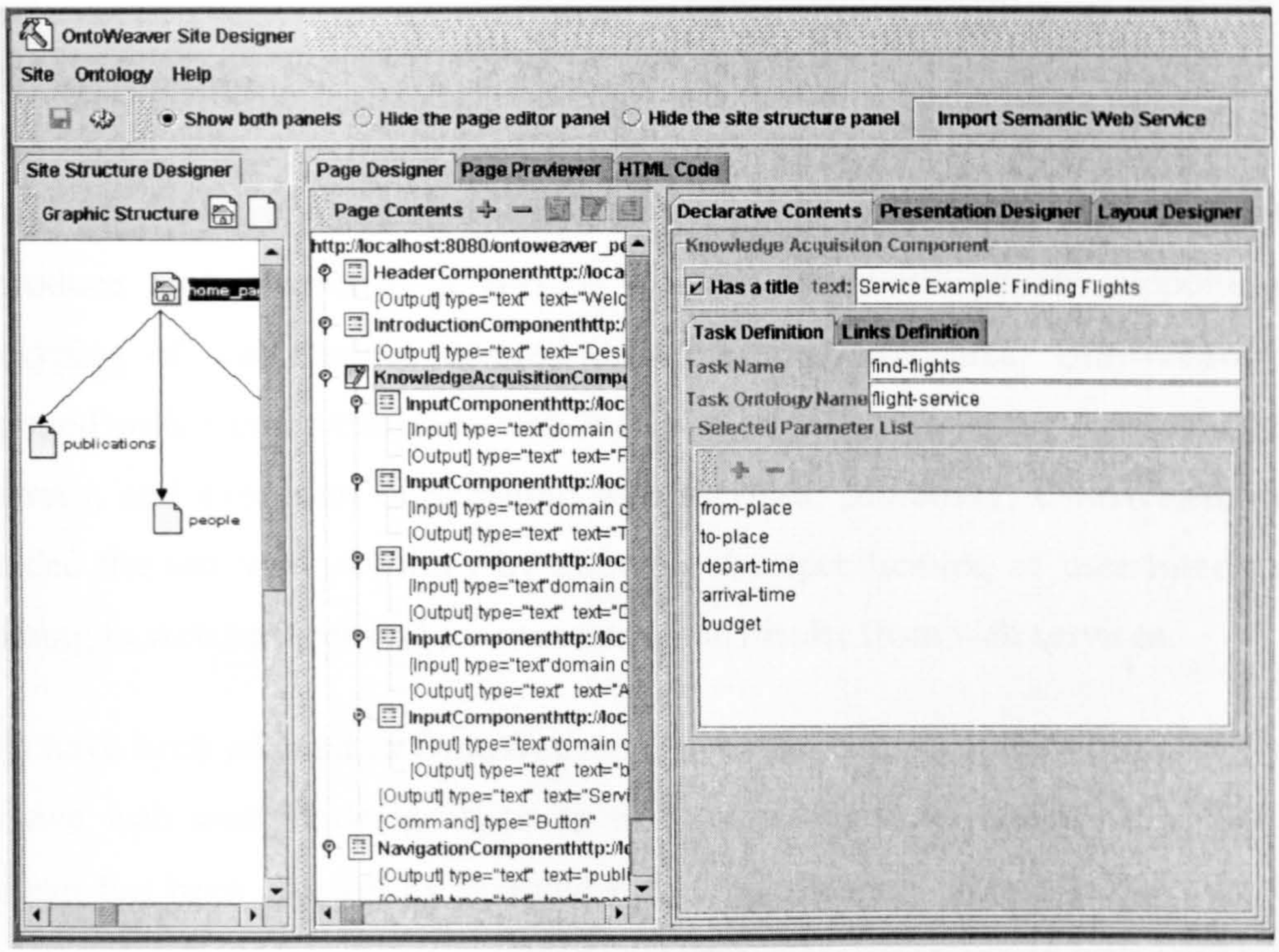


Figure 8.9 A screenshot of the OntoWeaver-S Site Designer. The pane on the far right side allows the specification of user interfaces for accessing web service by means of allowing the specification of the associated task and the input roles which need information from end users.

8.5 Related work

Current approaches to web site design typically model data-intensive web sites at three levels: *domain modelling*, *navigation modelling*, and *presentation modelling*. However, with the partial exception of WebML, none of these frameworks address the access to remote web services at a high level of abstraction. WebML relies on explicit site models to enable the design and development of data-intensive web applications. It has been extended recently by Brambillla et al. (2003) by means of a set of web service hypertext primitives for communicating with web services.

However, it is not clear how the extended framework addresses the problem of providing user interfaces for accessing web services. In contrast with this solution, OntoWeaver-S provides site view constructs to allow the explicit specification of web services within user interface components. Thus, OntoWeaver-S supports the integration of web services not only at the functional level (as WebML does) but also at the site view layer.

8.6 Conclusions

In this work we have augmented OntoWeaver, a *data-driven* web design framework, to produce OntoWeaver-S, a *service driven* framework which supports rapid prototyping of web service centred data-intensive web sites. OntoWeaver-S is integrated with a comprehensive web service platform, IRS-II, for the specification, discovery, and execution of semantic web services. Moreover, OntoWeaver-S has extended the site view ontology by allowing the specification of user interface for accessing to web services and for presenting the results from web services.

Tools have been adapted in OntoWeaver-S in order to support service centred data-intensive web sites at design time as well as at run time. Specifically, The Site Designer has been adapted to facilitate i) the specification of web services in target web sites and ii) the specification of user interfaces for accessing to web services and for presenting the results. Furthermore, a new tool called Service Integrator has been prototyped to provide support for the integration of web services into data-intensive web sites at run time. It gathers input information from a form which is associated with a web service, calls the web service platform to invoke the specified web service, and passes the results back.

OntoWeaver-S is the first toolkit which attempts to integrate web services into a high level design framework. Although we have taken OntoWeaver as our test bed to integrate web services, the method used in this chapter can also be used in other web site design frameworks.

Chapter 9: Contributions and future work

9.1 Contributions

This work is concerned with an approach which applies the notion of ontology to model all aspects of data-intensive web sites. By using ontologies, web site specification is formalized and can be shared during the life cycle of a web site, thus easing the job of web site maintenance and management. Furthermore, as the entire site model is available for reasoning, the scope of customization is not limited. Recommendations can be produced for web developers to improve their design by applying a set of critiquing rules to reason upon the entire site model. Finally, as the provided explicit shared semantics, the target web sites can be picked up by semantic-aware agents and thus benefit from intelligent services (e.g. indexing, searching, and customization) provided by third parties.

The main characteristics of OntoWeaver comprises i) using ontologies to drive the design and development of data-intensive web sites, ii) providing a comprehensive set of ontologies to support the specification of all aspects of web sites, and iii) providing a customization framework to support the specification of customization requirements at design time and the delivery of customization support at run time.

In addition to the site ontologies and the customization framework, we have also developed an OntoWeaver tool suite which facilitates different tasks involved in web site design and an OntoWeaver extension called OntoWeaver-S which addresses the issue of integration of web services with data-intensive web sites.

9.1.1 The site view ontology

The site view ontology provides fine-grained modelling support for user interfaces and navigation structures of the target web site. On the one hand, it provides comprehensive support for the specification of navigation structures. In particular,

the site view ontology addresses static navigation patterns where no additional information is involved in navigation as well as dynamic navigation patterns where contextual information flow is required. On the other hand, unlike current approaches which only address typical user interface elements of web pages, the site view ontology also addresses atomic user interface elements (e.g., input fields and command elements) which can not be further decomposed into sub elements, and composite user interface elements (e.g., web pages and generic components) which are typically composed of a number of sub elements. Thus, the site view ontology realizes a composition mechanism and allows web developers to express complex user interfaces. At the same time, the site view ontology covers all site view elements found in data-intensive web sites. It supports the specification of static user interface elements whose content is defined at design time, dynamic user interface elements whose content is retrieved from back-end data sources at run time, and interactive user interface elements which allow end users to type in information and invoke the associated services.

9.1.2 The presentation ontology

The presentation ontology provides explicit vocabularies to support the specification of presentation styles and layouts for web sites. On the one hand, the presentation ontology distinguishes different features of presentation styles of site view elements and proposes correspond constructs to support the specification. Specifically, it provides three constructs: one for the specification of generic presentation styles (e.g., backgrounds, colours, and fonts), one for the specification of presentation styles where widgets are involved (e.g., input elements, dynamic output elements, and command elements), and one for the specification of how to render dynamic content (e.g., how many columns in one page, and how many records one page displays).

On the other hand, the presentation ontology provides two main layout constructs to model organizations of site view elements. One is the construct *TextLayout* which models the layout of atomic site view elements in terms of *alignment*, specifying the alignment of a user interface element within the component that contains this

element The other is the construct *ComponentLayout* which abstracts the organization features of composite site view elements by dividing the layout of a composite site view elements into five sub areas (i.e. *top*, *left*, *middle*, *right*, and *bottom*) and allowing the position of the sub-elements into different areas. Each area can display a number of elements in a specified layout direction, i.e. horizontal direction or vertical direction. As composite site view components can further sub composite elements. The layout of each sub element can be further specified. Thus, the construct *ComponentLayout* supports the expression of complex layouts.

The presentation specification model in OntoWeaver is completely separate from the site view model. The separation is achieved by using the Uniform Resource Identifiers (URI) to identify the site view elements upon which the presentation object works. The separation offers a flexible way for developers to specify different visual appearances for web sites for different purposes, e.g., user groups and devices.

9.1.3 The customization framework

The customization framework addresses the limitation of current web modelling approaches on customization support by means of i) exploiting the advantage of the declarative specification of web sites, ii) separating the specification of customization requirements from other aspects of web sites, and iii) providing support for the specification of customization requirements at design time and for the execution of customization at run time. Firstly, as all user interface elements and their presentation instructions are specified declaratively, the entire site model is available to customization. Secondly, the customization framework separates the specification of customization from other aspects of the target web site, thus enabling the web site design process to be more flexible. Web developers no longer have to anticipate what can be customized at the stage of site view design, like they have to in approaches (e.g., WebML, WUML, and HERA). Finally, OntoWeaver provides comprehensive support for the specification of customization requirements. It offers a customization rule model to support the construction of customization rules. Thus,

web developers do not have to rely on ad-hoc approaches to augment individual's roles into site specifications.

9.1.4 The OntoWeaver tool suite

Different roles are involved in the design process of data-intensive web sites. As illustrated in chapter 7, the OntoWeaver tool suite supports these different design roles to achieve their tasks based upon the OntoWeaver modelling approach.

The architecture of the OntoWeaver tool suite comprises i) a knowledge warehouse which serves as an ontology repository storing all the ontologies involved in the design and management of web sites, ii) server-side service components which provide services for accessing to and manipulating data stored in the underlying knowledge warehouse, iii) an OntoWeaver Server which provides connections between front-end tools and the server-side components, iv) a set of design-time tools which support the design tasks (e.g., site structure design, site view design, presentation design, and customization design) of data-intensive web sites, and v) a set of run-time tools which support the target web sites at run time, including generating customized web pages and providing support for accessing and manipulating the underlying domain data.

9.1.5 OntoWeaver-S

OntoWeaver-S is an OntoWeaver extension, which supports rapid prototyping of web service centred data-intensive web sites. It is integrated with a comprehensive web service platform, IRS-II, for the specification, discovery, and execution of semantic web services. OntoWeaver-S has extended the site view ontology by allowing the specification of user interface for accessing to web services and for presenting the results from web services.

Tools have been adapted in OntoWeaver-S in order to support service centred data-intensive web sites at design time as well as at run time. Specifically, The Site Designer has been adapted to facilitate i) the specification of web services in target

web sites and ii) the specification of user interfaces for accessing to web services and for presenting the results. Furthermore, a new tool called Service Integrator has been prototyped to provide support for the integration of web services into data-intensive web sites at run time. It gathers input information from a form which is associated with a web service, calls the web service platform to invoke the specified web service, and passes the results back.

9.2 Open issues and future work

The conceptual web model approach described in the thesis focuses on the specification of data-intensive web sites, which is only one particular problem involved in the design and development of web sites. As a result, without additional research and development, the approach discussed here will only have a very limited impact on web developers. Therefore, it is appropriate to begin the discussion about outstanding issues by focusing on the problems which need to be overcome:

- Guidelines need to be provided, which provide methodological support on how to use the proposed approach to facilitate the specification of each aspect of target web sites.
- The issue of data integration should be addressed in future, which allows OntoWeaver to support the access and manipulation of the underlying heterogeneous domain data.
- More powerful critiquing facility needs to be provided, which supports the validation of complex site specifications and the specification of critiquing rules, thus offering customized critiquing service for web developers.
- More powerful customization facility needs to be provided, which i) offers comprehensive rule controlling strategy to handle situations when more than one rule can be applied and ii) exploits a number of customization techniques to capture and acquire user information, thus allowing the target web site to be more responsive to individual users.

- The usability of the OntoWeaver tool suite needs to be improved in order to provide more comprehensive support for the construction of data-intensive web sites.

9.2.1 Providing design guidelines

As illustrated in chapter 7, web site design in OntoWeaver involves a number of design steps, including i) requirements collection and analysis, ii) domain ontology design, iii) site structure design, iv) page content design, v) presentation design, and vi) customization design. Although there is a tool suite which offers comprehensive support for developers to achieve their tasks of most steps, no methodological support is available on how to use the OntoWeaver approach to achieve the task of each design step. As a result, web developers may find it difficult to achieve their tasks. Web design guidelines have been explicitly addressed in approaches like RMM and UWE. In future, we will try to produce appropriate design guidelines about how to use OntoWeaver to facilitate the design and development of data-intensive web sites by examining guidelines developed in both the field of conceptual web modelling and the field of ontology building and construction.

9.2.2 Handling heterogeneous domain data

As indicated in chapter 4, one major assumption we have made in this research is that the underlying heterogeneous data sources of a web site are integrated together in an agreed global schema (i.e. the domain ontology). As clarified in chapter 5, the OntoWeaver approach relies on the conceptual links between site views and the underlying domain ontology to realize dynamic access to domain data for web sites. Hence, as long as the agreed global schema is provided, the OntoWeaver approach is able to provide comprehensive support for the design and development of data-intensive web sites which present data sources in different representations by means of the global schema. However, OntoWeaver requires external technologies to site between the domain ontology and the heterogeneous data sources and solve the data integration issue. In particular, OntoWeaver prefers zero extra cost solutions (i.e. no extra effort from OntoWeaver).

In principle two kinds of data integration techniques have been developed, which are data warehousing [Hammer et al., 1995] and mediated approaches [Wiederhold, 1992]. While the data warehousing approaches extract and integrate data from diverse sources in advance, the mediated approaches perform integration at run time by e.g. querying data sources by means of the provided global schema. To make use of these techniques for providing support for data integration in OntoWeaver, we need to ensure i) the data presented in target web sites are always up-to-date, ii) the process of processing data integration is instant, and iii) minimal (zero or nearly zero) efforts are required from web developers. In future, we will examine data integration techniques and develop a solution which meets these requirements.

9.2.3 Providing more powerful critiquing facility

Web site design critiquing is an important functionality for web site design frameworks, which allows developers to gain feedback and recommendations over the design result and helps developers to improve the target web sites. At the moment, simple rules have been embedded within the OntoWeaver tools to support this functionality. More powerful critiquing facility could be provided by i) defining complex constraints to verify the validity of complex site specifications and ii) allowing the specification of critiquing rules which exploits the advantage of the declarative specification of web sites and offers customized critiquing services for web developers.

9.2.4 Providing more powerful customization facility

The customization framework in OntoWeaver has demonstrated the advantages of using an ontology-based approach to web site design with respect to the customization support. However, as described in chapter 6, one limitation of the OntoWeaver customization framework is that it does not provide comprehensive support for rule controlling. The customization framework groups customization actions into one rule when their customization conditions are the same. Thus, when one specific condition is satisfied, only one rule will be fired. However, there are

always situations where several conditions could be satisfied at the same time, thus more than one rule could be applied. In such circumstances, the customization engine relies on the physical order of customization rules stored to decide their priority. This strategy is far from comprehensive. In future work, we will investigate several possibilities to capture priorities for customization rules, e.g., automatic or semi-automatic discovery or manual acquisition.

Another limitation of the customization framework is that it does not provide mechanisms to capture the contextual information of users itself nor does it provide mechanisms to integrate with existing tools (e.g., machine leaning, data mining) to do so. The OntoWeaver customization framework currently only handles user profiles represented in the pre-defined user ontology, which means data coming from other customization tools cannot be used. This greatly limits the customization capability. In future we will take these issues into account and extend the customization framework to a more powerful one by providing mechanisms which allow external customization techniques for user information modelling and acquisition to be easily plugged in, thus allowing the target web site to be more responsive to individuals. For example, mechanisms could be provided which allow web developers to specify and plug particular customization techniques into their target web sites according to their domain specific requirements,

9.2.5 Improving usability

In this work we have built two data-intensive web sites by using the OntoWeaver approach and the OntoWeaver tool suite. However, during the design process, we have also observed that the layout design in OntoWeaver is not as flexible as in other What-You-See-Is-What-You-Get web site design environments. Although OntoWeaver does not attempt to replace web site design environments, the usability is important to OntoWeaver as its goal is to offer comprehensive support for web developers constructing data-intensive web sites. Hence, we need to address this issue in future work, e.g. by investing the integration of OntoWeaver with other web site design environments.

9.3 Outlook: modelling web sites on the semantic web

OntoWeaver is a conceptual web modelling approach, which applies the notion of ontology to model all aspects of data-intensive web sites. It provides fine-grained as well as coarse-grained support for the specification of all aspects of data-intensive web sites, including navigation structures, compositional structures of web pages, presentation styles, and layouts. Furthermore, OntoWeaver provides comprehensive support for customization design as well as customization delivery for data-intensive web sites.

The semantic web [Berners-Lee et al., 2001] is a vision of the next generation of the World Wide Web. It extends the current web by associating well-structured meaning with web resources. How to design web sites which fit in this vision is a challenge for web site design frameworks. A number of semantic web site initiatives, semantic web portals, have been built recently, which allow users enjoying the benefits gained from the semantic web technology on information sharing and exchanging in specified communities. Examples include OntoWeb portal [Spyns et al., 2002][1], Esperonto portal[2], MindSwap[3], KnowledgeWeb[4], and the KMi Semantic Portal[5]. These semantic web portals typically employ the domain ontology as a share basis for information communication and information exchanging.

As described earlier in chapter 5 (in section 5.5.4), there are a number of ontology-based tools developed, which address the generation of semantic web portals. Examples include OntoWebber [Jin et al., 2001], SEAL [Stojanovic et al., 2001], ODESeW [Corcho et al., 2003], and KAON [Volz et al., 2003]. These tools however either do not offer meta-models to support the specification (e.g., ODESeW and

---

[1] http://www.ontoweb.org/ (Accessed 24 June 2005)

[2] http://www.esperonto.net/ (Accessed 24 June 2005)

[3] http://www.mindswap.org/ (Accessed 24 June 2005)

[4] http://knowledgeweb.semanticweb.org/ (Accessed 24 June 2005)

[5] http://semanticweb.kmi.open.ac.uk/ (Accessed 24 June 2005)

KAON) or their meta-models are not comprehensive enough to support the specification of complex semantic web sites even if they provide explicit meta-models (e.g., OntoWebber). Furthermore, these approaches do not support the integration of web services with their target web sites.

Comparing to the tools mentioned above, OntoWeaver provides more comprehensive support for the design and development of semantic web sites, as it already employs semantic web technologies to benefit the web site design process. Specifically, it makes use of the semantic web standards RDF and RDF Schema as the underlying language to represent all aspects of the target web sites. Thus, the entire model of the target web application can be exploited by semantic-aware applications (e.g., semantic searching, index, and customization).

Furthermore, facilities can be easily added to OntoWeaver, which support the adding of semantic annotations to the target web sites. Specifically, static annotations can be pre-defined manually at design time. On the other hand, dynamic annotations can be generated from the underlying databases, as demonstrated in [Handschuh et al., 2003] and [Stojanovic et al., 2002], and be associated with web pages at run time. Thus, the target data-intensive web sites are semantically annotated and understandable by machines as well as human beings.

Finally, semantic navigation (which has been demonstrated in tools like Magpie [Domingue et al., 2004]) can be added, which brings not only data but also *knowledge* and knowledge services which are typically under the surface of data to user. Specifically, semantic links could be generated along with the generation of web pages, which allow user to navigation through not only just surface data but also deep knowledge. For example, when user browses information about a person in an academic organization web site, a number of semantic links could be presented allowing him/her to browse relevant information about this person (e.g., publications, projects). Such knowledge could come from the underlying databases or from the associated knowledge services.

References

Abrams, M., Phanouriou, C., Batongbacal, A. L., Williams, S. M., & Shuster, J. E. (1999), "UIML: An Appliance-Independent XML User Interface Language", in Proceedings of the 8th International World Wide Web Conference, May 11-14, 1999, Toronto, Canada, available from: http://www8.org/w8-papers/5b-hypertext-media/uiml/uiml.html (Accessed 24 June 2005).

Dey, A. K. & Abowd, G. D. (2001), "A Conceptual Framework and a Toolkit for Supporting the Rapid Prototyping of Context-Aware Applications", Human-Computer Interaction (HCI) Journal, Vol. 16 (2-4), 2001, pp. 97-166.

Ankolekar, A., Burstein, M., Hobbs, J., Lassila, O., Martin, D., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Payne, T., & Sycara, K. (2002), "DAML-S: Web Service Description for the Semantic Web", in Proceedings of the 1st International Semantic Web Conference, June 9-12, 2002, Sardinia, Italy, pp. 348-363.

Ardissono, L. & Goy, A. (1999), "Tailoring the Interaction with Users in Electronic Shops", in Proceedings of the 7th International Conference on User Modelling, June 20-24, 1999, Banff, Canada, pp. 35-44.

Atzeni, P., Mecca, G., & Merialdo, P. (1998), "Design and Maintenance of Data-Intensive Web Sites", in Proceeding of the 6th International Conference on Extending Database Technology (EDBT) (Lecture Notes in Computer Science Vol. 1377), March 23-27, 1998, Valencia, Spain, pp. 436-450.

Beaumont, I. (1994), "User Modelling in the Interactive Anatomy Tutoring System ANATOM-TUTOR", User Modelling and User-Adapted Interaction, Vol. 4 (1), pp. 21-45.

Berners-Lee, T., Fielding, R., & Masinter, L. (1998), "Uniform Resource Identifiers (URI): Generic Syntax", available from: http://www.ietf.org/rfc/rfc2396.txt (Accessed 24 June 2005).

Berners-Lee, T., Hendler, J., & Lassila, O. (2001), "The Semantic Web", Scientific American, May, 2001.

Beged-Dov, G., Brickley, D., Dornfest, R., Davis, I., Dodds, L., Eisenzopt, J., Galbraith, D., Guha, R.V., MacLeod, K., Miller, E., Swartz, A., & van der Vlist, E. (2000), "RDF Site Summary (RSS) 1.0", available from: http://groups.yahoo.com/group/rss-dev/files/specification.html (Accessed 24 June 2005).

Bochicchio, M., Paiano, R., & Paolini P. (1999), "JWeb: An HDM Environment for Fast Development of Web Applications", in Proceedings of IEEE International Conference on Multimedia Computing and Systems (ICMCS 1999), June 7-11, 1999, Florence, Italy, Volume 2, pp. 809 -813.

Borst, P., Akkermans, J., Pos, A., & TOP, J. (1995), "The PhysSys Ontology for Physical Systems", in Bredeweg, B., editor, Working Papers of the Ninth International Workshop on Qualitative Reasoning QR'95, pp. 11-21, University of Amsterdam.

Boyle, C. & Encarnacion, A. O. (1994), "MetaDoc: An Adaptive Hypertext Reading System", User Models and User Adapted Interaction, Vol. 4 (1), 1994, pp. 1-19.

Brambilla, M., Ceri, S., Comai, S., & Fraternali, P. (2003), "Model-driven Development of Web Services and Hypertext Applications", in Proceedings of the 7th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2003), July 27-30, 2003, Florida, USA, available from: http://www.webml.org/webml/upload/ent5/1/scii2003_fraternali.pdf (Accessed 24 June 2005).

Brusilovsky, P. (1992), "Intelligent Tutor, Environment and Manual for Introductory Programming", Educational and Training Technology International, Vol. 29 (1), pp. 26-34.

Brusilovsky, P. (1996), "Methods and techniques of adaptive hypermedia", User Modelling and User Adapted Interaction, Vol. 6 (2-3), 1996, pp. 87-129.

Brusilovsky, P., Schwarz, E., & Weber, G. (1996), "ELM-ART: An Intelligent Tutoring System on World Wide Web", in Frasson, C., Gauthier, G., & Lesgold, A. (Ed.), Intelligent Tutoring Systems (Lecture Notes in Computer Science Vol. 1086), pp. 261-269.

Brusilovsky, P., Eklund, J., & Schwarz, E. (1998), "Web-based Educations for All: A Tool for Development Adaptive Courseware", in Proceedings of the 7th International World Wide Web Conference, April 14-18, 1998, Brisbane, Australia, pp. 291-300.

Brusilovsky, P., & Cooper, D. W. (1999), "ADAPTS: Adaptive Hypermedia for a Web-based Performance Support System", in Proceedings of the 2nd Workshop on Adaptive Systems and User Modelling on the WWW, May 11-14, 1999, Toronto, Canada, pp. 41-47.

Brusilovsky, P. (2001), "Adaptive Hypermedia", User Modelling and User Adapted Interaction, Vol. 11 (1-2), 2001, pp. 87-110.

Cabral, L., Domingue, J., Motta, E., Payne, T., & Hakimpour, F. (2004), "Approaches to Semantic Web Services: An Overview and Comparisons", in Proceedings of the 1st European Semantic Web Symposium (ESWS 2004) (Lecture Notes in Computer Science Vol. 3053), May 10-12, 2004, Crete, Greece, pp. 225-239.

Cappi, J., Rossi, G., Fortier, A., & Schwabe, D. (2001), "Seamless Personalization of E-commerce Applications", in Proceedings of the International Workshop on E-commerce and conceptual Modelling, November 27-30, 2001, Yokohama, Japan, pp. 457-470.

Carro, R. M., Pulido, E. & Rodríguez, P. (1999), "TANGOW: Task-based Adaptive learNer Guidance on the WWW", in Proceedings of the 2nd Workshop on Adaptive Systems and User Modelling on the WWW, May 11-14, 1999, Toronto, Canada, pp. 49-57.

Ceri, S., Fraternali, P. & Paraboschi, S. (1999a), "Design Principles for Data-Intensive Web Sites", SIGMOD Record, Vol. 24 (1), March, 1999, pp. 84-89.

Ceri, S., Fraterali, P., & Paraboschi, S. (1999b), "Data-Driven One-To-One Web Site Generation for Data-Intensive Applications", in Proceedings of the 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, pp. 615-626.

Ceri, S., Fratenali, P., & Bongio, A. (2000), "Web Modelling Language (WebML): a Modelling Language for Designing Web Sites", in Proceedings of the 9th International World Wide Web Conference, May 15-19, 2000, Amsterdam, pp. 135-157.

Ceri, S., Fraterali, P. & Matera, M. (2002), "Conceptual Modelling of Data-Intensive Web Applications", IEEE Internet Computing, Vol. 6 (4), July-August, 2002, pp. 20-30.

Chen, P. (1976), "The Entity-Relationship Approach: Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1 (1), 1976, pp. 9-36.

Chin, D. N. (1989), "KNOME: Modelling What the User Knows in UC", in A. Kobsa and W. Wahlster, editors, User Models in Dialog Systems, pp. 74-107, Springer-Verlag, 1989, ISBN:0-387-18380-9.

Corcho, O., Gomez-Perez, A. Lopez-Cima, A. Lopez-Garcia, V., & Suarez-Figueroa, M. C. (2003), "ODESeW: Automatic Generation of Knowledge Portals for Intranets and Extranets", in Proceedings of the 2nd International Semantic Web Conference 2003 (ISWC 2003) (Lecture Notes in Computer Science Vol. 2870), October 20-23, 2003, Florida, USA, pp. 802-817.

Costagliola, G., Ferrucci, F., & Francese, R. (2002), "Web Engineering: Models and Methodologies for the Design of Hypermedia Applications", in Chang, S. K., editor, Handbook of Software Engineering & Knowledge Engineering, Vol. 2, Emerging Technologies, 2002, pp. 181-199, World Scientific.

Cowan, D. D. & Lucena, C. J. P. (1995), "Abstract Data Views, An Interface Specification Concept to Enhance Design for Reuse", IEEE Transactions on Software Engineering, Vol. 21 (3), March, 1995, pp. 229-243.

Da Silva, P. P. (2000), "User Interface Declarative Models and Development Environments: A Survey", in Proceedings of the 7th International Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'2000), June 5-6, 2000, Limerick, Ireland, pp. 207-226.

De Bra, P. & Calvi, L. (1998), "AHA! An Open Adaptive Hypermedia Architecture", The New Review of Hypermedia and Multimedia, Vol. 4, 1998, pp. 115-140.

De Bra, P., Brusilovsky, P., & Houben, G. (1999), "Adaptive Hypermedia: From Systems to Framework", ACM Computing Surveys, Vol. 31 (4), 1999, available from: http://www.cs.brown.edu/memex/ACM_HypertextTestbed/papers/25.html (Accessed 24 June 2005).

De Bra, P., Stash, N., & De Lange, B. (2003), "AHA! Adding Adaptive Behaviour to Websites", in Proceedings of the NLUUG Conference, May, 2003, the Netherlands, available from: http://www.win.tue.nl/~debra/nluug2003/nluug2003.pdf (Accessed 24 June 2005).

De Troyer, O. M. F. & Leune, C. J. (1998), "WSDM: a User Centered Design Method for Web Sites", in Proceedings of the 7th International World Wide Web Conference, April 14-18, 1998, Brisbane, Australia, pp. 85-94.

Diaz, A., Isakowitz, T., Maiorana, V., & Gilabert, G. (1995), "RMC: A tool to Designing WWW Applications", in Proceedings of the 4th International World Wide Web Conference, December 11-14, 1995, Boston, USA.

Domingue, J. B., & Dzbor, M. (2004), "Magpie: Browsing and Navigating on the Semantic Web", in Proceedings of the International Conference on Intelligent User Interfaces, January 13-16, 2004, Madeira, Portugal, pp. 191-197.

Eriksson, H., Puerta, A. R., & Musen, M. A. (1994), "Generation of Knowledge-Acquisition Tools from Domain Ontologies", International Journal of Human-Computer Studies, Vol. 41, 1994, pp. 425-453.

Fensel, D. & Bussler, C. (2002), "The Web Service Modelling Framework WSMF", available from: http://informatik.uibk.ac.at/users/c70385/wese/wsmf.bis2002.pdf (Accessed 24 June 2005).

Fernandez, M., Florescu, D., Kang, J., Levy, A., & Suciu, D. (1997), "STRUDEL: A Web Site Management System", in Proceedings of the ACM SIGMOD International Conference on Management of data, May 13-15, 1997, Tucson, Arizona, USA, pp. 549-552.

Fernandez, M., Florescu, D., Levy, A., & Suciu, D. (1998), "Catching the Boat with Strudel: Experiences with a Web-site Management System", in Proceedings of the 1998 ACM SIGMOD international conference on Management of data, June 2-4, 1998, Seattle, Washington, USA, pp. 414-425, ACM Press.

Fink, J. & Kobsa, A. (2000), "A Review and Analysis of Commercial User Modelling Servers for Personalization on the World Wide Web", User Modelling and User-Adapted Interaction, Vol. 10 (3-4), 2000, pp. 209-249.

Fink, J., Kobsa, A., & Nill, A. (1998), "Adaptable and Adaptive Information Provision for All Users, Including Disabled and Elderly People", The New Review of Hypermedia and multimedia, Vol. 4, pp. 163-188.

Frasincar, F., Houben, G., & Vdovjak, R. (2002), "Specification Framework for Engineering Adaptive Web Applications", in Proceedings of the 11th International World Wide Web Conference, May 7-11, 2002, Honolulu, Hawaii, USA, available

from: http://wwwconf.ecs.soton.ac.uk/archive/00000225/01/index.html (Accessed 24 June 2005).

Fraternali, P. & Paolini, P. (1998), "A Conceptual Model and a Tool Environment for Developing More Scalable Dynamic Applications", in Proceedings of the 6th International Conference on Extending Database Technology, March 23-27, 1998, Valencia, Spain, pp. 421-435.

Fraternali, P. (1999), "Tools and Approaches for Developing Data-Intensive Web Applications: a Survey", ACM Computing Surveys, Vol. 31 (3), September, 1999, pp. 227-263.

Garzotto, F., Paolini, P. & Schwabe, D. (1993), "HDM—A Model-Based Approach to Hypertext Application Design", ACM Transactions on Information Systems, Vol. 11 (1), January, 1993, pp. 1-26.

Gomez, J., Cachero, C., & Pastor, O. (2000), "Extending a Conceptual Modelling Approach to Web Application Design", in Proceedings of the 12th International Conference on Advanced Information Systems Engineering (Lecture Notes in Computer Science Vol. 1789), June 5-9, 2000, Stockholm, Sweden, pp. 79-93.

Good, M. D., Whiteside, J. A., Wixon, D. R., & JonesBuilding, S. J. (1984), "Building a User-Derived Interface", Communications of the ACM, Vol. 27 (10), October, 1984, pp. 1032-1043.

Grosso, W. E., Eriksson, H., Fergerson, R. W., Gennari, J. H., Tu, S. W., & Musen, M. A. (1999), "Knowledge Modelling at the Millennium", in Proceedings of the 12th International Workshop on Knowledge Acquisition, Modelling and Management (KAW'99), October 16-21, 1999, Banff, Canada, available from: http://sern.ucalgary.ca/KSI/KAW/KAW99/papers/Grosso1/kmatm.pdf (Accessed 24 June 2005).

Gruber, T.R. (1993), "Toward Principles for the Design of Ontologies Used for Knowledge Sharing", in Formal Ontology in Conceptual Analysis and Knowledge Representation, edited by Nicola Guarino and Roberto Poli, Kluwer Academic Publishers. Substantial revision of paper presented at the International Workshop on Formal Ontology, March, 1993, Padova, Italy.

Halasz, F. & Schewartz, M. (1994), "The Dexter Hypertext Reference Model", Communications of ACM, Vol. 37 (2), February, 1994, pp. 30-39.

Hammer, J., Garcia-Molina, H., Labio, W., Widom, J., & Zhuge, Y. (1995), "The Stanford Data Warehousing Project", IEEE Data Engineering Bulletin, Vol. 18 (2), June, 1995, pp. 41-48.

Handschuh, S., Staab, S., & Volz, R. (2003), "On Deep Annotation", in Proceedings of the 12th International World Wide Web Conference (WWW2003), May 20-24, 2003, Budapest, Hungary, available from: http://www.aifb.uni-karlsruhe.de/WBS/sha/papers/p273_handschuh.pdf (Accessed 24 June 2005).

Hennicker, R. & Koch, N. (2000), "A UML-based Methodology for Hypermedia Design", in Proceedings of the 3rd International Conference on the Unified Modelling Language (UML 2000), October, 2000, York, England, pp. 410-424.

Henze, N. & Nejdl, W. (1999), "Adaptivity in the KBS Hyperbook System", in Proceedings of the 2nd Workshop on Adaptive Systems and User Modelling on the WWW, May 11-14, 1999, Toronto, Canada, pp. 67-74.

Hook, K. (1996), "A Glass Box Approach to Adaptive Hypermedia", PhD thesis, Department of Computer and Systems Sciences, Stockholm University.

Hohl, H., Böcker, H., & Gunzenhäuser, R. (1996), "HYPADAPTER: An Adaptive Hypertext System for Exploratory Learning and Programming", User Modelling and User-Adapted Interaction, Vol. 6 (2-3), pp. 131-155.

IBM (2002), "IBM Web Services Toolkit – A Showcase for Emerging Web Services Technologies", available from: http://www-3.ibm.com/software/solutions/webservices/wstk-info.html (Accessed 24 June 2005).

Isakowitz, T., Kamis, A., & Koufaris, M. (1997), "Extending RMM: Russian Dolls and Hypertext", in Proceedings of the Hawaii International Conference on System Sciences (HICSS-30), January 7 - 10, 1997, Maui, Hawaii, available from: http://citeseer.ist.psu.edu/cache/papers/cs/3391/http:zSzzSzrmm-java.stern.nyu.eduzSzrmmzSzpaperszSzdolls.pdf/isakowitz96extending.pdf (Accessed 24 June 2005).

Isakowitz, T., Stohr, E. A., & Balasubramaninan, P. (1995), "RMM: A Methodology for Structured Hypermedia Design", Communications of the ACM, Vol. 38 (8), 1995, pp. 34-44.

Jenamani, M., Mohapatra, P. K. j., & Ghose, S. (2002), "Online Customized Index Synthesis in Commercial Web Sites", IEEE Intelligent Systems, Vol. 17 (6), pp. 20-26.

Jin, Y., Decker, S., & Wiederhold, G. (2001), "OntoWebber: Model-Driven Ontology-Based Web site Management", in Proceedings of the International Semantic Web Working Symposium, July 30 – August 1, 2001, Stanford, California, available from: http://www.semanticweb.org/SWWS/program/full/paper55.pdf (Accessed 24 June 2005).

Kappel, G., Retschitzegger, W., & Schwinger, W. (2000), "Modelling Customizable Web Applications A Requirement's Perspective", in Proceedings of the International Conference on Digital Libraries, November 13-16, 2000, Kyoto, Japan, available at: http://www.dsic.upv.es/~west/iwwost01/files/contributions/UniversityLinz/ICDL2000.pdf (Accessed 24 June 2005).

Kappel, G., Retschitzegger, W., Pöll, B., & Schwinger, W. (2001), "Modelling Ubiquitous Web Applications – The WUML Approach", in Proceedings of the International Workshop on Data Semantics in Web Information Systems (DASWIS 2001), November, 2001, Yokohama, Japan, pp. 183-197.

Kappel, G., Kimmerstorfer, E., Hofer, T., Pröll, B. Retschitzegger, W., & Schwinger, W. (2002), "Towards a Generic Customisation Model for Ubiquitous Web Applications", in Proceedings of the 2nd International Workshop on Web Oriented Software Technology, July 10, 2002, Malaga, Spain, pp. 79-104.

Kappel, G., Pröll, B., Retschitzegger, W., & Schwinger, W. (2003), "Customisation for Ubiquitous Web Applications - A Comparison of Approaches", International Journal of Web Engineering and Technology (IJWET), Vol. 1 (1), 2003, pp. 79-111.

Kay, J. & Kummerfeld, R. J. (1994), "An Individualised Course for the C Programming Language", in Proceedings of the 2nd World Wide Web Conference, October 17-20, 1994, Chicago, USA, available from: http://archive.ncsa.uiuc.edu/SDG/IT94/Proceedings/Educ/kummerfeld/kummerfeld.html (Accessed 24 June 2005).

Klapsing, R., Neumann, G., & Conen, W. (2001), "Semantics in Web Engineering: Applying the Resource Description Framework", IEEE MultiMedia Journal, Vol. 8 (2), April-June, 2001, pp. 62-68.

Kobsa, A. (1993), "User Modeling: Recent Work, Prospects and Hazards", in Schneider-Hufschmidt, M, Kuehme, T, and Malinowski, U (Eds.) Adaptive User Interfaces: Principles and Practice North-Holland, the Netherlands (1993), available from: http://www.ics.uci.edu/~kobsa/papers/1993-aui-kobsa.pdf (Accessed 24 June 2005).

Kobsa, A., Müller, D., & Nill, A. (1994), "KN-AHS: An Adaptive Hypertext Client of the User Modeling System BGP-MS", in Proceedings of the 4th International Conference on User Modelling, August 15-19, Hyannis, Massachusetts, pp. 99-105. Reprinted in: M. T. Marbury and W. Wahlster, eds. (1998): Intelligent User Interfaces. San Mateo, CA: Morgan Kaufman, pp. 372-378.

Kobsa, A., Koenemann, J., & Pohl, W. (2001), "Personalized Hypermedia Presentation Techniques for Improving Online Customer Relationships", The Knowledge Engineering Review, Vol. 16 (2), 2001, pp. 111-155.

Koch, N. (1999), "A Comparative Study of Methods for Hypermedia Development", Technical Report 9905, Ludwig-Maximilians-University Munchen, November 1999, available from: http://www.pst.informatik.uni-muenchen.de/personen/kochn/techrep/hypdev.pdf (Accessed 24 June 2005).

Koch, N. (2001), "Software Engineering for Adaptive Hypermedia Applications", PhD thesis, Reihe Softwaretechnik 12, Uni-Druck Publishing Company, Munich, 2001, available from: http://www.pst.informatik.uni-muenchen.de/personen/kochn/PhDThesisNoraKoch.pdf (Accessed 24 June 2005).

Koch, N., Kraus, A., & Hennicker, R. (2001), "The Authoring Process of the UML-based Web Engineering Approach", in Proceedings of the 1st International Workshop on Web-oriented Software Technology, June 18-20, 2001, Valencia, Spain, available from: http://www.dsic.upv.es/~west/iwwost01/files/contributions/NoraKoch/Uwe.pdf (Accessed 24 June 2005).

Landay, J. A. & Borriello, G. (2003), "Design Patterns for Ubiquitous Computing", IEEE computer, Vol. 36 (8), 2003, pp. 93-95.

Langley, P. (1999), "User Modelling in Adaptive Interfaces", in Proceedings of the 7th International Conference on User Modelling, June 20-24, 1999, Banff, Canada, pp. 357 -370.

Manber, U., Patel, A., & Robison, J. (2000), "Experience with Personalization on Yahoo!", in Communications of the ACM, Vol. 43 (8), 2000, pp. 35-39.

Maurino, A. & Fraternali, P. (2002), "Commercial Tools for the Development of Personalized Web Applications: a Survey", in Proceedings of the Third conference on Electronic Commerce and Web Technologies (EC-WEB 2002), September 2-6, 2002, Aix-en-Provance, France, pp. 99-109.

Menczer, F., Street, W. N., & Monge, A. E. (2002), "Adaptive Assistants for Customized E-shopping", IEEE Intelligent Systems, Vol. 17 (6), pp. 12-19.

Morville, P. & Rosenfeld, L. (1998), "Information Architecture for the World Wide Web", O'Reilly, ISBN 1-56592-282-4.

Mostafa, J., (2002), "Guest Editor's Introduction: Information Customization", IEEE Intelligent Systems, Vol. 17 (6), pp. 8-11.

Motta, E. (1999), "Reusable Components of Knowledge Modelling: Case Studies in Parametric Design Problem Solving", IOS Press, Amsterdam, 1999.

Motta, E., Shum, S. B., & Domingue, J. (2000), "Ontology-Driven Document Enrichment: Principles, Tools and Applications", International Journal Human-Computer Studies (2000), Vol. 52 (5), pp. 1071-1109.

Motta, E., Domingue, J., Cabral, L., & Gaspari, M. (2003), "IRS-II: A Framework and Infrastructure for Semantic Web Services", in Proceedings of the 2nd International Semantic Web Conference 2003 (ISWC 2003), October 20-23, 2003, Sanibel Island, Florida, USA, pp. 306-318.

Mulvenna, M. D., Anand, S. S., & Bucher, A.G. (2000), "Personalization on the Net Using Web Mining: Introduction", Communications of ACM, Vol. 43 (8), 2000, pp. 121-125.

Murugesan, S., Desshpande, Y., Hansen, S., & Ginige, A. (1999), "Web Engineering: A New Discipline for Development of Web-based Systems", in Proceedings of the 1st ICSE workshop on Web Engineering, May 16-17, 1999, Los Angeles, USA, available from: http://www-itec.uni-klu.ac.at/~harald/proseminar/web11 (Accessed 24 June 2005)

Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R.,Senator, T., & Swartout W.R. (1991), "Enabling technology for knowledge sharing", AI Magazine, Vol. 12 (3), pp. 36-56.

Nejdl, W. & Wolpers, M. (1999), "KBS Hyperbook - a Data-Driven Information System on the Web", in Proceedings of the 8th World Wide Web Conference, May 11-14, 1999, Toronto, Canada, available from: http://www.kbs.uni-hannover.de/Arbeiten/Publikationen/1999/www8/ (Accessed 24 June 2005).

OASIS (2002), "UDDI Specification", available from: http://www.uddi.org/specification.html (Accessed on 24 June 2005).

Perkowitz, M. & Etzioni, O. (2000), "Towards Adaptive Web Sites: Conceptual Framework and Case Study". Artificial Intelligence, Vol. 118, 2000, pp. 245-275.

Puerta, A. (1997), "A Model-Based Interface Development Environment", IEEE Software, Vol. 14 (4), 1997, pp. 40-47.

Puerta, A. & Eisenstein, J. (2002), "XIML: A Common Representation for Interaction Data", in Proceedings of the 7th International Conference on Intelligent User Interfaces, January 12-15, 2002, Miami, Florida, USA, pp. 214-215.

Retschitzegger, W. & Schwinger, W. (2000), "Towards Modelling of DataWeb Applications - A Requirement's Perspective", in Proceedings of the Americas Conferenc on Information Systems (AMCIS), August, 2000, Long Beach, California, Vol. I, pp. 149-155.

Rossi, G., Schwabe, D., & Guimarães, R. (2001), "Designing Personalized Web Applications", in Proceedings of the 10th International World Wide Web Conference, May 1-5, 2001, Hong Kong, pp. 275-284.

Schwabe, D. & Rossi, G. (1998), "An Object Oriented Approach to Web-Based Application Design", Theory and Practice of Object Systems, Vol. 4 (4), 1998, pp. 201-225, Wiley and Sons, New York, ISSN 1074-3224.

Schwabe, D., Pontes, R. A., & Moura, I. (1999), "OOHDM-Web: An Environment for Implementation of Hypermedia Applications in the WWW", SigWEB Newsletter, Vol. 8 (2), June, 1999.

Schwabe, D., Guimaraes, R. M., & Rossi, G. (2002), "Cohesive Design of Personalized Web Applications", IEEE Internet Computing, Vol. 6 (2), March-April, 2002, pp. 34-43.

Shardanand, U. & Maes, P. (1995), "Social Information Filtering: Algorithms for Automating 'Word of Mouth'", in Proceedings of the ACM SIGCHI Conference on Human Factors in Computing Systems, May 7-11, 1995, Colorado, USA, pp. 210-217.

Spyns, P., Oberle, D., Volz, R., Zheng, J., Jarrar, M., Sure, Y., Studer, R., & Meersman, R. (2002), "OntoWeb - A Semantic Web Community Portal", in Proceedings of the 4th International Conference on Practical Aspects of Knowledge Management (PAKM), December 2-3, 2002, Vienna, Austria, pp. 189-200.

Stojanovic, N., Maedche, A., Staab, S., & Studer, R. (2001), "SEAL – A Framework for Developing SEmantic PortALs", in Proceedings of the 1st International Conference on Knowledge Capture (KCAP-2001), October 21-23, 2001, Victoria, B.C., Canada, pp. 1-22.

Stojanovic, L., Stojanovic, N., & Volz, R. (2002), "Migrating Data-Intensive Web Sites into the Semantic Web", in Proceedings of the 17th ACM Symposium on Applied Computing (SAC 2002), March 10-14, 2002, Madrid, Spain, pp. 1100-1107, ACM Press.

SUN (2003), "Java Web Services Developer Pack", available from: http://java.sun.com/webservices/webservicespack.html (Accessed 24 June 2005).

Szekely, P. (1996), "Retrospective and Challenges for Model-Based Interface Development", in Proceedings of the 2nd International Workshop on Computer-Aided Design of User Interfaces (CADUI'96), June, 1996, Eurographics, Belgium, pp. 1-27.

Szekely, P., Sukaviriya, P., Castells, P., Muthukumarasamy, J., & Salcher, E. (1995), "Declarative Interface Models for User Interface Construction Tools: the MASTERMIND Approach", in Proceedings of the IFIP Working Conference on Engineering for Human-Computer Interaction, August 14-18, 1995, Wyoming, USA, pp. 120-150.

Volz, R., Oberle, D., Staab, S., & Motik, B. (2003), "KAON SERVER - A Semantic Web Management System", in Alternate Track Proceedings of the 12th International World Wide Web Conference (WWW2003), May 20-24, 2003, Budapest, Hungary,

available from: http://www2003.org/cdrom/papers/alternate/P029/p29-volz.html (Accessed 24 June 2005).

Wiederhold, G. (1992), "Mediators in the Architecture of Future Information Systems", IEEE computer, Vol. 25 (3), March, 1992, pp. 38-49.

W3C (1999a), "Resource Description Framework (RDF) Model and Syntax", available from: http://www.w3.org/TR/PR-rdf-syntax/ (Accessed 24 June 2005).

W3C (1999b), "PIDL - Personalized Information Description Language", W3C Note, 1999, available from: http://www.w3.org/TR/NOTE-PIDL (Accessed 24 June 2005).

W3C (2000a), "Resource Description Framework (RDF) Schema Specification 1.0", W3C Candidate Recommendation, available from: http://www.w3.org/TR/2000/CR-rdf-schema-20000327/ (Accessed 24 June 2005).

W3C (2000b), "Simple Object Access Protocol (SOAP) (2000)", W3C Note 08, available from: http://www.w3.org/TR/SOAP/ (Accessed 24 June 2005).

W3C (2001), "Web Services Description Language (WSDL) (2001)", W3C Note 15, available from: http://www.w3.org/TR/wsdl (Accessed 24 June 2005).

APPENDIX A: The site view ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
   ]>

<rdf:RDF xml:lang="en"
  xmlns     = "http://kmi.open.ac.uk/people/yuangui/siteviewontology#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs ="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd  = "http://www.w3.org/2001/XMLSchema#" >

<!-- Classes -->

<rdfs:Class rdf:ID="Site">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>Site</rdfs:label>
   <rdfs:comment>
      This construct describes a web site as a collection of logical resources.
  </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="SiteResource">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>SiteResource</rdfs:label>
   <rdfs:comment>
      This construct models web pages.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ResourceComponent">
  <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>resource component</rdfs:label>
   <rdfs:comment>
      This construct describes components of web pages.
  </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="DataComponent">
  <rdfs:subClassOf rdf:resource="#ResourceComponent"/>
   <rdfs:label>Data Component</rdfs:label>
   <rdfs:comment>
      This construct expresses those components which present dynamic data content
that is retrieved from the underlying domain knowledge base or obtained from the
specified web service.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="KAComponent">
   <rdfs:subClassOf rdf:resource="#ResourceComponent"/>
   <rdfs:label>Knowledge Acquisition Component</rdfs:label>
   <rdfs:comment>
      This construct describes those components which allow end users to input
information for creating a new instance for the specified class entity or for
invoking the specified web service.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="SearchComponent">
   <rdfs:subClassOf rdf:resource="#ResourceComponent"/>
   <rdfs:label>Search Component</rdfs:label>
   <rdfs:comment>
```

```
      The construct defines the components which allow end users to make queries over
the underlying knowledge base.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="InputComponent">
   <rdfs:subClassOf rdf:resource="#ResourceComponent"/>
   <rdfs:label>Input Component</rdfs:label>
   <rdfs:comment>
      This construct describes those components which gather input from end users for
a particular slot of the specified class entity or for a particular input role of the
specified web service. An input component typically contains an output element
presenting an explanation about the corresponding input field and an input element
presenting the input field.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="OutputComponent">
   <rdfs:subClassOf rdf:resource="#ResoureComponent"/>
   <rdfs:label>Output Component</rdfs:label>
   <rdfs:comment>
      This construct describes the components which present content for a particular
slot of the specified class entity or for a particular output role of the specified
web service. An output component typically contains an output element presenting an
explanation about the dynamic values and a dynamic output element presenting dynamic
values.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Input">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>Input</rdfs:label>
   <rdfs:comment>
      This construct describes the atomic elements which present input fields to
allow the gathering of information from end users for a particular slot of the
specified class entity or for a particular input role of the specified web service.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Output">
   <rdfs:subClassOf rdf:resource="#Output"/>
   <rdfs:label>Output</rdfs:label>
   <rdfs:comment>
      This construct describes the atomic elements which present static information.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="DynamicOutput">
   <rdfs:subClassOf rdf:resource="#Output"/>
   <rdfs:label>Dynamic Output</rdfs:label>
   <rdfs:comment>
      This construct describes the atomic elements which present dynamic content for
a particular slot of the specified class entity or for a particular output role of
the specified web service.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Command">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>Command</rdfs:label>
   <rdfs:comment>
      This construct describes the atomic elements which enable end users to invoke
the specified services and bring dynamic content to users.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="LinkItem">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>LinkItem</rdfs:label>
   <rdfs:comment>
      This construct describes the link relationships between web pages.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="DynamicLinkItem">
```

```
   <rdfs:subClassOf rdf:resource="#LinkItem"/>
   <rdfs:label>dynamic link</rdfs:label>
   <rdfs:comment>
   This construct describes dynamic links which is retrieved from the underlying
knowledge base or obtained from the associated web service.
</rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ContextualLinkItem">
   <rdfs:subClassOf rdf:resource="#LinkItem"/>
   <rdfs:label>contextual links</rdfs:label>
   <rdfs:comment>
      This construct describes link items through which contextual information flows
when navigation from the source page to the linked web page.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="MetaData">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>meta data</rdfs:label>
   <rdfs:comment>
      This construct describes meta-data for web pages.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="InstanceConstraint">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>instance constraint</rdfs:label>
   <rdfs:comment>
      This construct describes constraints of instances.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Task">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes tasks associated with data components, knowledge
acquisition components, and search components.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="RelationOperator">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes relation operators, e.g., EQUAL etc.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="LogicOperator">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes logic operators, e.g., NOT, AND, and OR.
   </rdfs:comment>
</rdfs:Class>


<!-- Properties -->

<!-- Properties for the construct Site -->
<rdf:Property rdf:ID="hasIndexResource">
   <rdfs:domain rdf:resource="#Site"/>
   <rdfs:range rdf:resource="#SiteResource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasResource">
   <rdfs:domain rdf:resource="#Site"/>
   <rdfs:range rdf:resource="#SiteResource"/>
</rdf:Property>

<!-- Properties for the construct SiteResource -->
<rdf:Property rdf:ID="hasComponent">
   <rdfs:domain rdf:resource="#SiteResource"/>
   <rdfs:domain rdf:resource="#ResourceComponent"/>
   <rdfs:range rdf:resource="#ResourceComponent"/>
```

```
</rdf:Property>

<rdf:Property rdf:ID="hasMetaData">
   <rdfs:domain rdf:resource="#SiteResource"/>
   <rdfs:range rdf:resource="#MetaData"/>
</rdf:Property>

<!-- Properties for the construct ResourceComponent -->
<rdf:Property rdf:ID="hasInput">
   <rdfs:domain rdf:resource="#ResourceComponent"/>
   <rdfs:range rdf:resource="#Input"/>
</rdf:Property>

<rdf:Property rdf:ID="hasOutput">
   <rdfs:domain rdf:resource="#ResourceComponent"/>
   <rdfs:range rdf:resource="#Output"/>
</rdf:Property>

<rdf:Property rdf:ID="hasCommand">
   <rdfs:domain rdf:resource="#ResourceComponent"/>
   <rdfs:range rdf:resource="#Command"/>
</rdf:Property>

<rdf:Property rdf:ID="hasInstanceConstraint">
   <rdfs:domain rdf:resource="#DataComponent"/>
   <rdfs:domain rdf:resource="#ContextualLinkItem"/>
   <rdfs:range rdf:resource="#InstanceConstraint"/>
</rdf:Property>

<rdf:Property rdf:ID="hasClassEntity">
   <rdfs:domain rdf:resource="#DataComponent"/>
   <rdfs:domain rdf:resource="#KAComponent"/>
   <rdfs:domain rdf:resource="#SearchComponent"/>
   <rdfs:domain rdf:resource="#DynamicOutput"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasSlotEntity">
   <rdfs:domain rdf:resource="#DynamicOutput"/>
   <rdfs:domain rdf:resource="#DynamicLinkItem"/>
   <rdfs:domain rdf:resource="#Input"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<!-- Properties for the construct Output -->
<rdf:Property rdf:ID="hasOutputValueType">
   <rdfs:domain rdf:resource="#Output"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasOutputValue">
   <rdfs:domain rdf:resource="#Output"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasLinkItem">
   <rdfs:domain rdf:resource="#Output"/>
   <rdfs:range rdf:resource="#LinkItem"/>
</rdf:Property>

<rdf:Property rdf:ID="hasAssociatedResourceURI">
   <rdfs:domain rdf:resource="#LinkItem"/>
   <rdfs:domain rdf:resource="#Command"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasCommandText">
   <rdfs:domain rdf:resource="#Command"/>
   <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Property>

<rdf:Property rdf:ID="hasConstrainedClassEntity">
   <rdfs:domain rdf:resource="#InstanceConstraint"/>
```

```
    <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasConstrainedSlotEntity">
    <rdfs:domain rdf:resource="#InstanceConstraint"/>
    <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasConstrainedRelation">
    <rdfs:domain rdf:resource="#InstanceConstraint"/>
    <rdfs:range rdf:resource="#RelationOperator"/>
</rdf:Property>

<rdf:Property rdf:ID="hasConstrainedValue">
    <rdfs:domain rdf:resource="#InstanceConstraint"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasLogicOperator">
    <rdfs:domain rdf:resource="#InstanceConstraint"/>
    <rdfs:range rdf:resource="#LogicOperator"/>
</rdf:Property>

<!-- Properties for the construct MetaData -->
<rdf:Property rdf:ID="hasPageTitle">
    <rdfs:domain rdf:resource="#MetaData"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasAssociatedClassEntity">
    <rdfs:domain rdf:resource="#MetaData"/>
    <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasIntroduction">
    <rdfs:domain rdf:resource="#MetaData"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasDescription">
    <rdfs:domain rdf:resource="#MetaData"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasAuthor">
    <rdfs:domain rdf:resource="#MetaData"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<!-- Properties for the construct Task -->
<rdf:Property rdf:ID="hasTask">
    <rdfs:domain rdf:resource="#Command"/>
    <rdfs:domain rdf:resource="#KAComponent"/>
    <rdfs:domain rdf:resource="#DataComponent"/>
    <rdfs:range rdf:resource="Task"/>
</rdf:Property>

<rdf:Property rdf:ID="hasTaskDescription">
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="taskName">
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="taskOntologyName">
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="inputRole">
```

```
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:domain rdf:resource="#Input"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="outputRole">
    <rdfs:domain rdf:resource="#Task"/>
    <rdfs:domain rdf:resource="#DynamicOutput"/>
    <rdfs:domain rdf:resource="#DynamicLinkItem"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<!-- Primitives for the construct Task (built-in functionalities) -->
<rdf:Description rdf:about="#NEW-DATA-ENTRY" >
  <rdf:type rdf:resource="#Task" />
</rdf:Description>

<rdf:Description rdf:about="#DATA-QUERYING">
  <rdf:type rdf:resource="#Task" />
</rdf:Description>

<rdf:Description rdf:about="#DATA-UPDATING">
  <rdf:type rdf:resource="#Task" />
</rdf:Description>

<!-- Primitives for the construct RelationOperator -->
<rdf:Description rdf:about="#EQUAL">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#NOT-EQUAL">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#GREATER-THAN">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#NO-LESS-THAN">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#LESS-THAN">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#NO-GREATER-THAN">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<rdf:Description rdf:about="#PART-OF">
  <rdf:type rdf:resource="#RelationOperator" />
</rdf:Description>

<!-- Primitives for the construct LogicOperator -->
<rdf:Description rdf:about="#AND">
  <rdf:type rdf:resource="#LogicOperator" />
</rdf:Description>

<rdf:Description rdf:about="#OR">
  <rdf:type rdf:resource="#LogicOperator" />
</rdf:Description>

<rdf:Description rdf:about="#NOT">
  <rdf:type rdf:resource="#LogicOperator" />
</rdf:Description>
</rdf:RDF>
```

APPENDIX B: The presentation ontology

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
   ]>

<rdf:RDF xml:lang="en"
 xmlns="http://kmi.open.ac.uk/people/yuangui/sitepresentationontology#"
 xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
 xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" >

<!-- Classes -->

<rdfs:Class rdf:ID="SitePresentation" >
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes presentation models for the specified site view model.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Presentation">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct defines presentation style for site view components.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Layout">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct defines layouts for site view components.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="TextLayout">
   <rdfs:subClassOf rdf:resource="#Layout"/>
   <rdfs:comment>
      This construct defines layouts for atomic site view components.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ComponentLayout">
   <rdfs:subClassOf rdf:resource="#Layout"/>
   <rdfs:comment>
      This construct defines layouts for composite site view components.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="ComponentAreaLayout">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct abstracts organizations of a sub area for a composite site view
component.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="TextAlignment">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
     This construct describes the alignment of text, e.g., left, center, right, etc.
   </rdfs:comment>
</rdfs:Class>
```

```
<rdfs:Class rdf:ID="AreaSize">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
        This construct describes the size of a sub area for a composite site view
component.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="AreaSizeType">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
        This construct describes the type of the specified size.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="PresentationTemplate">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes general presentation styles of site view elements.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="WidgetPresentationTemplate">
   <rdfs:subClassOf rdf:resource="#PresentationTemplate"/>
   <rdfs:comment>
        This construct abstracts presentation styles for site view components which
are involved in with widgets, e.g. dynamic output elements, command elements, and
input elements.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="DataComponentPresentationTemplate">
   <rdfs:subClassOf rdf:resource="#PresentationTemplate"/>
   <rdfs:comment>
        This construct abstracts presentation styles for data components which present
dynamic data content.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Font">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
        This construct abstracts fonts.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="AreaLayoutDirection">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
     This construct describes the direction (e.g., horizon or vertical) by which the
associated site view elements are placed in a sub area of a composite site view
element.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="FontStyle">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<rdfs:Class rdf:ID="WidgetType">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
</rdfs:Class>

<!--  Properties   -->

<!-- Properties for the construct SitePresentation -->
<rdf:Property rdf:ID="hasSiteURI">
   <rdfs:domain rdf:resource="#SitePresentation"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasSiteEntityURI">
   <rdfs:domain rdf:resource="#SitePresentation"/>
   <rdfs:domain rdf:resource="#Presentation"/>
```

```
    <rdfs:domain rdf:resource="#Layout"/>
    <rdfs:domain rdf:resource="#ComponentAreaLayout"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasPresentation">
    <rdfs:domain rdf:resource="#SitePresentation"/>
    <rdfs:range rdf:resource="#Presentation"/>
</rdf:Property>

<rdf:Property rdf:ID="hasLayout">
    <rdfs:domain rdf:resource="#SitePresentation"/>
    <rdfs:range rdf:resource="#Layout"/>
</rdf:Property>

<rdf:Property rdf:ID="hasTemplate">
    <rdfs:domain rdf:resource="#SitePresentation"/>
    <rdfs:range rdf:resource="#PresentationTemplate"/>
</rdf:Property>

<!-- Properties for the construct Presentation -->
<rdf:Property rdf:ID="hasTemplateURI">
    <rdfs:domain rdf:resource="#Presentation"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<!-- Properties for the construct PresentationTemplate -->
<rdf:Property rdf:ID="hasBackgroundColor">
    <rdfs:domain rdf:resource="#PresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasBackgroundImage">
    <rdfs:domain rdf:resource="#PresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasForeColor">
    <rdfs:domain rdf:resource="#PresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>
<rdf:Property rdf:ID="hasFont">
    <rdfs:domain rdf:resource="#PresentationTemplate"/>
    <rdfs:range rdf:resource="#Font"/>
</rdf:Property>

<!-- Properties for the construct Font -->
<rdf:Property rdf:ID="hasFontFamily">
    <rdfs:domain rdf:resource="#Font"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasFontSize">
    <rdfs:domain rdf:resource="#Font"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasFontStyle">
    <rdfs:domain rdf:resource="#Font"/>
    <rdfs:range rdf:resource="#FontStyle"/>
</rdf:Property>

<!-- Properties for the construct WidgetPresentationTemplate -->
<rdf:Property rdf:ID="hasWidgetType">
    <rdfs:domain rdf:resource="#WidgetPresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<!-- Properties for the construct DataComponentPresentationTemplate -->
<rdf:Property rdf:ID="hasColumnCount">
    <rdfs:domain rdf:resource="#DataComponentPresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
    <rdfs:comment>
```

```
      This construct describes the number of columns for presenting data objects.
    </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="hasRowsPerPage">
    <rdfs:domain rdf:resource="#DataComponentPresentationTemplate"/>
    <rdfs:range rdf:resource="&xsd;String"/>
    <rdfs:comment>
        This construct describes the number of rows for presenting data objects per
web page.
    </rdfs:comment>
  </rdf:Property>

<!-- Properties for the construct ComponentLayout -->
  <rdf:Property rdf:ID="hasTopAreaLayout">
    <rdfs:domain rdf:resource="#ComponentLayout"/>
    <rdfs:range rdf:resource="#ComponentAreaLayout"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasLeftAreaLayout">
    <rdfs:domain rdf:resource="#ComponentLayout"/>
    <rdfs:range rdf:resource="#ComponentAreaLayout"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasMiddleAreaLayout">
    <rdfs:domain rdf:resource="#ComponentLayout"/>
    <rdfs:range rdf:resource="#ComponentAreaLayout"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasRightAreaLayout">
    <rdfs:domain rdf:resource="#ComponentLayout"/>
    <rdfs:range rdf:resource="#ComponentAreaLayout"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasBottomAreaLayout">
    <rdfs:domain rdf:resource="#ComponentLayout"/>
    <rdfs:range rdf:resource="#ComponentAreaLayout"/>
  </rdf:Property>

<!-- Properties for the construct ComponentAreaLayout -->
  <rdf:Property rdf:ID="hasLayoutDirection">
    <rdfs:domain rdf:resource="#ComponentAreaLayout"/>
    <rdfs:range rdf:resource="#AreaLayoutDirection"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasAreaSize">
    <rdfs:domain rdf:resource="#ComponentAreaLayout"/>
    <rdfs:range rdf:resource="#AreaSize"/>
  </rdf:Property>

  <!-- Properties for the construct AreaSize -->
  <rdf:Property rdf:ID="hasSizeType">
    <rdfs:domain rdf:resource="#AreaSize"/>
    <rdfs:range rdf:resource="AreaSizeType"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasWidth">
    <rdfs:domain rdf:resource="#AreaSize"/>
    <rdfs:range rdf:resource="&xsd;String"/>
  </rdf:Property>

  <rdf:Property rdf:ID="hasHeight">
    <rdfs:domain rdf:resource="#AreaSize"/>
    <rdfs:range rdf:resource="&xsd;String"/>
  </rdf:Property>

  <!-- Properties for the construct TextAlignment -->
  <rdf:Property rdf:ID="hasAlignment">
    <rdfs:domain rdf:resource="#TextLayout"/>
    <rdfs:range rdf:resource="#TextAligment"/>
  </rdf:Property>

<!—Primitives -->
```

```
<!-- Primitives for the construct TextLayout -->
 <rdf:Description rdf:about="#LEFT" >
    <rdf:type rdf:resource="#TextAligment" />
 </rdf:Description>

  <rdf:Description rdf:about="#CENTER" >
     <rdf:type rdf:resource="#TextAligment" />
  </rdf:Description>

  <rdf:Description rdf:about="#RIGHT" >
     <rdf:type rdf:resource="#TextAligment" />
  </rdf:Description>

  <rdf:Description rdf:about="#TOP" >
     <rdf:type rdf:resource="#TextAligment" />
  </rdf:Description>

  <rdf:Description rdf:about="#MIDDLE" >
     <rdf:type rdf:resource="#TextAligment" />
  </rdf:Description>

  <rdf:Description rdf:about="#BOTTOM" >
     <rdf:type rdf:resource="#TextAligment" />
  </rdf:Description>

  <!-- Primitives for the construct FontStyle -->
  <rdf:Description rdf:about="#BOLD" >
     <rdf:type rdf:resource="#FontStyle" />
  </rdf:Description>

  <rdf:Description rdf:about="#ITALIC" >
     <rdf:type rdf:resource="#FontStyle" />
  </rdf:Description>

  <!-- Primitives for the construct WidgetType -->
  <rdf:Description rdf:about="#NONE-WIDGET" >
     <rdf:type rdf:resource="#WidgetType" />
  </rdf:Description>

  <rdf:Description rdf:about="#TEXTFIELD" >
     <rdf:type rdf:resource="#WidgetType" />
  </rdf:Description>

  <rdf:Description rdf:about="#TEXTAREA" >
     <rdf:type rdf:resource="#WidgetType" />
  </rdf:Description>

  <rdf:Description rdf:about="#BUTTON" >
     <rdf:type rdf:resource="#WidgetType" />
  </rdf:Description>

  <rdf:Description rdf:about="#IMAGE" >
     <rdf:type rdf:resource="#WidgetType" />
  </rdf:Description>

  <!-- Primitives for the construct AreaSizeType -->
  <rdf:Description rdf:about="#RELATIVE" >
     <rdf:type rdf:resource="#AreaSizeType" />
  </rdf:Description>

  <rdf:Description rdf:about="#ABSOLUTE" >
     <rdf:type rdf:resource="#AreaSizeType" />
  </rdf:Description>

  <!-- Primitives for the construct AreaLayoutDirection -->
  <rdf:Description rdf:about="#HORIZONTAL-DIRECTION" >
     <rdf:type rdf:resource="#AreaLayoutDirection" />
  </rdf:Description>

  <rdf:Description rdf:about="#VERTICAL-DIRECTION" >
     <rdf:type rdf:resource="#AreaLayoutDirection" />
  </rdf:Description>
</rdf:RDF>
```

APPENDIX C: The customization rule model

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
     <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
     <!ENTITY xsd  "http://www.w3.org/2001/XMLSchema#" >
   ]>

<rdf:RDF xml:lang="en"
  xmlns="http://kmi.open.ac.uk/people/yuangui/sitecustomizationontology#"
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:svo="http://kmi.open.ac.uk/people/yuangui/siteviewontology >


<!-- Classes -->

<rdfs:Class rdf:ID="CustomizationRule">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct abstracts customization rules.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="CustomizationCondition">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct defines conditions for customization rules.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="CustomizationAction">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct defines actions for customization rules.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Modification">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes details of how to customize the specified site view
elements.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="Condition">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:label>parameter clause</rdfs:label>
   <rdfs:comment>
      This construct allows the formalization of complex condition.
   </rdfs:comment>
</rdfs:Class>

<rdfs:Class rdf:ID="CustomizationActionType">
   <rdfs:subClassOf rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This construct describes the types of customization actions, e.g. modifying
site view elements, creating new site view elements, hiding site view elements, or
associating presentation instructions.
   </rdfs:comment>
</rdfs:Class>

<!--  Properties -->
<!-- Properties for the construct CustomizationRule -->
```

```
<rdf:Property rdf:ID="hasCustomizationCondition">
   <rdfs:domain rdf:resource="#CustomizationRule"/>
   <rdfs:range rdf:resource="#CustomizationCondition"/>
</rdf:Property>

<rdf:Property rdf:ID="hasCustomizationAction">
   <rdfs:domain rdf:resource="#CustomizationRule"/>
   <rdfs:range rdf:resource="#CustomizationAction"/>
</rdf:Property>

<!-- Properties for the construct CustomizationCondition -->
<rdf:Property rdf:ID="hasCondition">
   <rdfs:domain rdf:resource="#CustomizationCondition"/>
   <rdfs:range rdf:resource="#Condition"/>
</rdf:Property>

<!-- Properties for the construct CustomizationAction -->
<rdf:Property rdf:ID="hasSiteEntityURI">
   <rdfs:domain rdf:resource="#CustomizationAction"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasParentSiteEntityURI">
   <rdfs:domain rdf:resource="#CustomizationAction"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasCustomizationActionType">
   <rdfs:domain rdf:resource="#CustomizationAction"/>
   <rdfs:range rdf:resource="#CustomizationActionType"/>
</rdf:Property>

<rdf:Property rdf:ID="hasSiteEntityType">
   <rdfs:domain rdf:resource="#CustomizationAction"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
   <rdfs:comment>
      This property describes the type (e.g. outputs, inputs and commands) for the
newly created site view elements in terms of the site view ontology.
   </rdfs:comment>
</rdf:Property>

<rdf:Property rdf:ID="hasModification">
   <rdfs:domain rdf:resource="#CustomizationAction"/>
   <rdfs:range rdf:resource="#Modification"/>
</rdf:Property>

<!-- Properties for the construct Modification -->
<rdf:Property rdf:ID="hasSlotEntity">
   <rdfs:domain rdf:resource="#Modification"/>
   <rdfs:domain rdf:resource="#Condition"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasNewValue">
   <rdfs:domain rdf:resource="#Modification"/>
   <rdfs:range rdf:resource="&xsd;String"/>
   <rdfs:comment>
      This property describes the new value for the specified slot.
   </rdfs:comment>
</rdf:Property>

<!-- Properties for the construct Condition -->
<rdf:Property rdf:ID="hasClassEntity">
   <rdfs:domain rdf:resource="#Condition"/>
   <rdfs:range rdf:resource="&rdfs;Resource"/>
</rdf:Property>

<rdf:Property rdf:ID="hasRelationOperator">
   <rdfs:domain rdf:resource="#Condition"/>
   <rdfs:range rdf:resource="&svo;RelationOperator"/>
</rdf:Property>

<rdf:Property rdf:ID="hasSpecifiedValue">
```

```
   <rdfs:domain rdf:resource="#Condition"/>
   <rdfs:range rdf:resource="&xsd;String"/>
</rdf:Property>

<rdf:Property rdf:ID="hasLogicOperator">
   <rdfs:domain rdf:resource="#Condition"/>
   <rdfs:range rdf:resource="&svo;LogicOperator"/>
</rdf:Property>

<!-- Primitives -->

<!-- Primitives for the construct CustomizationActionType -->
<rdf:Description rdf:about="#MODIFY" >
  <rdf:type rdf:resource="#CustomizationActionType" />
</rdf:Description>

<rdf:Description rdf:about="#NEW" >
  <rdf:type rdf:resource="#CustomizationActionType" />
</rdf:Description>

<rdf:Description rdf:about="#HIDE" >
  <rdf:type rdf:resource="#CustomizationActionType" />
</rdf:Description>

<rdf:Description rdf:about="#PRESENTATION" >
  <rdf:type rdf:resource="#CustomizationActionType" />
</rdf:Description>

</rdf:RDF>
```